

LOMAS DATA PRODUCTS INC.

THUNDER 186 OWNER'S MANUAL

REV.0

PROPRIETARY NOTICE

The information and design of the equipment described herein are the sole property of LOMAS DATA PRODUCTS, INC. This document is proprietary to LOMAS DATA PRODUCTS, INC. and the information contained herein is for use by LOMAS DATA PRODUCTS, INC., its authorized representatives, and by users of this equipment. The information is not to be reproduced in whole or in part without written permission of an authorized official of LOMAS DATA PRODUCTS, INC.

LOMAS DATA PRODUCTS, INC.
66 HOPKINTON ROAD
WESTBOROUGH, MA 01581
617 366-6434

1.0 THUNDER 186

1.1 Introduction

The THUNDER 186 is based upon the newest in a family of 16 bit microprocessors from Intel. The 80186 is an advanced, high performance microprocessor with specially optimized capabilities for use in highly integrated systems. THUNDER 186 is a complete single board computer on a single height S100 bus board. It contains all components to form a high performance 16 bit computer without need of additional boards. The 80186 is upward compatible with 8086 and 8088 software. This insures a wide variety of software will be available as most of the popular personal computers use the the 8088 microprocessor.

1.2 Features

THUNDER 186 offers many features which are summarized below:

- * 80186 advanced microprocessor (8086/8088 compatible)
- * Addresses up to 1 Megabyte of memory
- * Full object code compatibility with 8086/8088 code
- * Integrated Interrupt controller with 11 vectored interrupts
- * 2 EPROM sockets for up to 64K bytes of EPROM
- * Strict conformance to IEEE 696
- * Floppy disk controller controls both 8" and 5 1/4" disk drives simultaneously
- * Two serial ports with software programmable baud rates up to 9600 baud
- * Centronics compatible parallel printer port
- * Software programmable real time interrupt
- * Two DMA channels, both accessible by bus peripherals

1.3 Limited Warranty Information

LOMAS DATA PRODUCTS will repair or replace, at our option, any parts found to be defective in either materials or workmanship for a period of one year from date of invoice. Defective parts must be returned to LOMAS DATA PRODUCTS for replacement.

If a defective part causes a LOMAS DATA PRODUCTS product to operate improperly during the one year warranty period, we will service it free (original owner only) if delivered and shipped at owner's expense to and from LOMAS DATA PRODUCTS. If improper operation is due to an error or errors on part of the purchaser, there may be a repair charge. The purchaser will be notified of any anticipated charges.

We are not responsible for damages caused by use of solder intended for purposes other than electronic equipment construction, failure to follow printed instructions, misuse or abuse, unauthorized modifications, theft, fire, accidents, or use

of our products in applications in other than those intended by LOMAS DATA PRODUCTS.

2.2 Word Size

Return to the purchaser of a fully functioning unit meeting all advertised specifications in effect as of the date of purchase is considered to be complete fulfillment of all warranty obligations assumed by LOMAS DATA PRODUCTS. This warranty covers only products marketed by LOMAS DATA PRODUCTS and does not cover other equipment used in conjunction with products from LOMAS DATA PRODUCTS.

Fastest Instruction

Prices and specifications are subject to change without notice, owing to the volatile nature and pricing structure of the electronics industry.

2.3 Memory Capacity

2.3.1 ROM/EPROM

On board: 64K bytes of EPROM maximum
8K if 2732's
beginning address F0000H
16K if 2764's beginning address FC000H
32K if 27128's beginning address F8000H
64K if 27256's beginning address F0000H

2.3.2 RAM

256K bytes of Dynamic RAM. 150 ns. 4184's allow full speed operation with wait states only on refresh collisions. Beginning address 0000H.

Offboard expansion: Up to 704 Kbytes of RAM, ROM or EPROM.

2.4 Processor Bus

THUNDER 186 is fully IEEE S100 bus compatible. Strict conformance to timing has been designed into the THUNDER 186.

2.5 Environmental Requirements

Temperature 0 to 50 C (32 to 122 F)
Relative humidity from 5% to 90% non-condensing

2.6 Physical Characteristics

Width 10 inches
Height 5 inches
Component height max .5 inches

2.7 Power Requirements

+8V at 2.5 amps typical 3.0 amps max.
 +16V and -16V at 100 ma.

2.8 Bus Pin Usage

TABLE 2.9

PIN #	SIGNAL	DESCRIPTION
1	8 VOLTS	I Instantaneous minimum greater than 7 volts. Instantaneous maximum less than 25 volts. Average maximum less than 11 volts.
2	+16 VOLTS	I Minimum 14.5 volts Maximum 21.5 volts. Instantaneous peak less than 35 volts.
3	XRDY	I One of two ready inputs from slave devices. Both must be true to continue.
4	VIO*(O.C.)	I Vectored interrupt 0 through 7 with interrupt 0 being the highest priority. VIO thru VI7 may be configured thru jumpers to cause vectored interrupts on any of the four 80186 vectored interrupt pins INTO thru INT3
5	VI1*(O.C.)	
6	VI2*(O.C.)	
7	VI3*(O.C.)	
8	VI4*(O.C.)	
9	VI5*(O.C.)	
10	VI6*(O.C.)	
11	VI7*(O.C.)	I Non-maskable interrupt.
12	NMI*(O.C.)	
13	Not implemented	
14	Not implemented	
15	A18	O Address bit 18.
16	A16	O Address bit 16.
17	A17	O Address bit 17.
18	SDBS*(O.C.)	I This signal disables the 8 status lines sM1, SMEMR, sINP, sOUT, sWO*, sINTA, sHLTA and sXTRQ*.
19	CDSB*(O.C.)	I This signal disables the five control out signals pSYNC, pSTVAL, pDBIN, pWR*, and pHLDA.
20	GND	Logic ground common with pins 100, 70, 53 and 50.
21	Not defined	
22	ADSB*(O.C.)	I This signal disables the 24 address lines A0 through A23.
23	DODSB*(O.C.)	I This signal disables the Data Out output buffers from D00 through D07.
24	0	O This master system clock derives from the processor clock.
25	pSTVAL*	O Status Valid Strobe indicating that address and status are valid during the current bus cycle.

THUNDER 186 MANUAL REV 0.0 (04-16-84)

26	pHLDA	0 Indicates to a requesting temporary master that the LIGHTNING ONE is relinquishing the bus.
27	Not used	
28	Not used	
29	A5	0 Address bit 5
30	A4	0 Address bit 4
31	A3	0 Address bit 3
32	A15	0 Address bit 15
33	A12	0 Address bit 12
34	A9	0 Address bit 9
35	DO1	O/I Data out bit 1
36	DO0	O/I Data out bit 0
37	A10	0 Address bit 10
38	DO4	O/I Data out bit 4
39	DO5	O/I Data out bit 5
40	DO6	O/I Data out bit 6
41	DI2	O/I Data in bit 2
42	DI3	O/I Data in bit 3
43	DI7	O/I Data in bit 7
44	SM1	0 This status line indicates that the current cycle is an instruction fetch cycle.
45	aOUT	0 This status line indicates that the current bus cycle is outputting data to an I/O port.
46	aINP	0 This status line indicates that the current bus cycle is an input cycle from an I/O port.
47	aMEMR	0 This status line indicates that the current bus cycle is a memory read.
48	aHLTA	0 Acknowledges that a HALT instruction has been executed.
49	clock	0 2 MHz asynchronous clock
50	GND	Logic ground common with Pins 100, 70, 53 and 20.
51	+8 volts	I Common with Pin 1.
52	-16 volts	I Instantaneous maximum -14.5 volts Instantaneous minimum -35 volts Average minimum greater than -21.5 volts
53	GND	Common with Pins 100, 70, 50 and 20.
54	SLAVE CLR*(O.C.)	0 A reset signal to bus slaves
55	Not implemented	
56	Not implemented	
57	Not implemented	
58	sXTRQ*	0 Requests 16 bit data transfer
59	A19	0 Address Bit 19
60	Not implemented	
61	A20	0 Address Bit 20
62	A21	0 Address Bit 21
63	A22	0 Address Bit 22

THUNDER 186 MANUAL REV 0.0 (04-16-84)

64	A23	O Address bit 23
65	DMAREQ0	I Channel 0 off board DMA request
66	DMAREQ1	I Channel 1 off board DMA request
67	PHANTOM*(O.C.)	O Disables normal slaves and enables PHANTOM slaves
68	MWRT	O Memory write pulse
69	Not used	
70	GND	Common with Pins 100, 53, 50, and 20
71	Not used	
72	RDY(O.C.)	I See Pin 3
73	not used	
74	HOLD*(O.C.)	I Used to request the permanent master to release the bus.
75	RESET*(O.C.)	O/I Used to reset bus master devices
76	pSYNC	O A control signal indicating BS1
77	pWR*	O Indicates valid data on the data out bus
78	pDBIN	O Requests that the selected slave place data on the DI bus lines
79	A0	O Address bit 0
80	A1	O Address bit 1
81	A2	O Address bit 2
82	A6	O Address bit 6
83	A7	O Address bit 7
84	A8	O Address bit 8
85	A13	O Address bit 13
86	A14	O Address bit 14
87	A11	O Address bit 11
88	DO2	O/I Data out bit 2
89	DO3	O/I Data out bit 3
90	DO7	O/I Data out bit 7
91	DI4	O/I Data in bit 4
92	DI5	O/I Data in bit 5
93	DI6	O/I Data in bit 6
94	DI1	O/I Data in bit 1
95	DIO	O/I Data in bit 0
96	aINTA	O Status signal indicated that the current bus cycle is an interrupt acknowledge cycle.
97	aWO*	O Indicates the current bus cycle transfers data from a bus master to a bus slave
98	Not used	
99	POC*	O Power on clear. Remains low for 10 msec. after power on.
100	GND	

3.0 CONFIGURING THE THUNDER 186

3.1 Introduction

THUNDER 186 provides a number of jumper options for user flexibility. The options provided by jumpering are as follows:

- EPROM selection
- Write precompensation (125 of 250 ns.)
- Slave reset connected to Master reset

3.2 EPROM SELECTION

Various EPROM types may be used in the THUNDER 186 CPU. The type used is selected by various jumper selections on JP1. The following table provides the jumper selections for the various EPROM types:

EPROM	JUMPERS(JP1)
2732	2-3,5-6
2764	2-3,5-6
27128	1-2,5-6
27256	1-2,4-5

3.3 FLOPPY DRIVE PRECOMPENSATION

The floppy disk controller provides write precompensation. The amount of precompensation is jumper selectable with jumper JP2. Pins 1 to 2 selects 125 nanosecond precompensation and pins 2 to 3 select 250 nanosecond precompensation. Normally 125 nanosecond precompensation is sufficient.

3.4 SLAVE RESET

The IEEE 696 specification for the S100 bus defines three different reset lines on the bus, POC(power on clear), RESET, and SLAVE CLR. THUNDER 186 asserts all three reset lines for 50 milliseconds after power on. Normally the front panel reset button only asserts RESET when pushed, leaving any bus slave devices in the previous condition. By inserting jumper JP3, both RESET and SLAVE CLR are asserted when the front panel reset button is pushed. This is the preferred position.

3.5 SERIAL PORT CONNECTORS

The serial port connectors (J1 and J2) are set up to use a direct one to one cable when connecting to a terminal. The following is the connector pin as seen at the rear panel of the computer when connected with a standard ribbon cable with a header at one end and a DB25S connector at the other end:

THUNDER 186 MANUAL REV 0.0 (04-16-84)

PIN NUMBER	SIGNAL	
2	RXDATA	
3	TXDATA	IN
4	DSR(DATA SET READY)	OUT
5	DTR(DATA TERMINAL READY)	IN
6	RTS(REQUEST TO SEND)	OUT
7	SIGNAL GROUND	OUT
20	CLEAR TO SEND	IN

CONNECTOR J1 IS THE CONNECTOR FOR YOUR TERMINAL. THE ONBOARD SERIAL MONITOR EXPECTS A TERMINAL CONNECTED TO THIS PORT. THUNDER IS SHIPPED WITH THIS PORT SET AT A BAUD RATE OF 9600 BAUD.

3.6 PARALLEL PRINTER PORT PIN ASSIGNMENTS

The parallel printer port (J3) has the following pin assignment when connected to the rear panel with LOMAS DATA PRODUCTS cable, part number 030-00026-01:

PIN NUMBER	SIGNAL	DIRECTION
1	STROBE	OUT
2	DATA 0	OUT
3	DATA 1	OUT
4	DATA 2	OUT
5	DATA 3	OUT
6	DATA 4	OUT
7	DATA 5	OUT
8	DATA 6	OUT
9	DATA 7	OUT
10	ACK	IN
11	PTR BUSY	IN
18	GND	
19	GND	
20	GND	
21	GND	
22	GND	
23	GND	

3.6 FLOPPY DISK DRIVE CONNECTOR PIN ASSIGNMENTS

8 INCH DRIVES(J5)		5 1/4 INCH DRIVES(J4)	
PIN	SIGNAL	PIN	SIGNAL
2	LOW CURRENT	2	TWO SIDED
4	FAULT RESET	4	NC
6	FAULT	6	DS3
8	NC	8	INDEX
10	TWO SIDED	10	DS0
12	NC	12	DS1
14	SIDE SELECT	14	DS2
16	NC	16	MOTOR ON
18	HEAD LOAD	18	DIRECTION
20	INDEX	20	STEP
22	READY	22	WRITE DATA
24	NC	24	WRITE ENABLE
26	DS0	26	TRACK 0
28	DS1	28	WRITE PROTECT
30	DS2	30	READ DATA
32	DS3	32	SIDE SELECT
34	DIRECTION	34	READY
36	NC		
38	WRITE DATA		
40	WRITE ENABLE		
42	TRACK 0		
44	WRITE PROTECT		
46	READ DATA		
48	NC		
50	NC		

ALL ODD PIN ARE GROUND ON BOTH CONNECTORS

4.0 PROGRAMMING THUNDER 186

THUNDER 186 comes with CONCURRENT CP/M-86 and the source code for the XIOS. The XIOS should be consulted as the primary source for programming the THUNDER 186. In addition to the XIOS source code the appendix includes a reprint of an application note on the 80186 microprocessor. This note contains detailed information about the internal features and programming of the 80186. The file, T186.LIB on the distribution diskette, contains equates for THUNDER 186 which provide all I/O addresses necessary to program THUNDER 186. Also the source code for the THUNDER 186 monitor is provided on the distribution diskette. This should provide useful for interfacing to the monitor and as further examples of programming the 80186 and the peripheral I.C.'s of the THUNDER 186.

5.0 THE MONITOR

5.1 Introduction

The monitor has the following commands:

B	Boot
C	Convert decimal to Hex
D	Dump memory
E	Enter data
F	Fill memory
G	Go
H	Hex Arithmetic +,-,/,*
I	Input from I/O port
L	Load Disk sectors to memory
M	Move memory
O	Output data to I/O port
R	Display Registers
S	Search memory for data string
T	Trace Instruction cycles
V	Verify memory
W	Write memory to disk

In addition to the above commands, the monitor includes power up diagnostics to insure the correct functioning of the required hardware. At power up time and each time the CPU is reset memory diagnostics that test the first 1K bytes of memory are executed. 1K bytes of memory starting at memory location 0 are the minimum amount of memory necessary to execute the monitor. If a memory error is encountered or if no memory is found the following message is printed at the console:

Memory failure or no memory present

Disk problems may be isolated with the disk utilities in the monitor. These basic tests will insure that the minimum requirements of the monitor are present. If an error is found, an attempt to report it is made using the console device. Execution continues even if an error is found.

5.2 Monitor Parameters

The monitor commands described in the following sections use combinations of the following parameters:

BYTE	2 Digit HEX number
WORD	4 digit HEX number
ADDRESS	5 digit HEX number
RANGE	Is either (ADDRESS) (ADDRESS) or (ADDRESS) L (WORD)
LIST	Is a series of byte values which may

STRING

be expressed as a sequence of BYTES separated by either a ',' or a SPACE, or as a STRING.

Is a number of characters enclosed either in (') or ("). The opening and closing quotes must be the same.

5.3 BOOT

B

B ADDRESS . . . Address

Switch 4

ON

code

stack

disk

00500

00400

8"

DD found

OFF

28000

400

5 1/4"

DD
512 bytes

Boot may take either of the above two forms. The first form loads the first two tracks from a single density diskette into memory starting at location 400H and then jumps to the first location. The Stack Pointer is set to 7000H. The second form loads the first two tracks as in the previous example and then sets breakpoints at the specified addresses. The tracks are loaded before the breakpoints are set so any errors in the breakpoints will not be found until after the load.

5.4 Convert

C word

Converts a four digit decimal number to its hex value and outputs the result to the console.

5.5 Dump

D ADDRESS

D RANGE

Dump displays the memory area specified as a parameter in the command as hex bytes followed by their ASCII representation. Nonprinting characters are shown as a '.' in the dump. The first form displays 128 bytes beginning at the specified address. The second form displays the whole range, which must be less than 64K bytes.

5.6 Enter

E ADDRESS,LIST

E ADDRESS

The first form enters the parameters in LIST into memory beginning at the specified address and then returns to command mode. The second form puts the monitor into enter mode, starting at the specified. After a carriage return, the address and its contents should be changed, type the new value to be entered. If no change is required, type a space and the next location may be examined and changed if required. A '--' will back up to the previous location. A carriage return will terminate the command and return to command mode.

5.7 Fill F RANGE,LIST

The specified address range is filled with the data specified in LIST. If the range is longer than the LIST, the LIST is repeated. The LIST may be just one byte which will cause the entire range to be filled with the same value.

5.8 Go G G ADDRESS . . . ADDRESS

Go sets all the registers from the register save area, including the CS and IP registers. Therefore, you must set the CS and IP registers to point to the program you wish to execute before you execute this command. The first form will not set any breakpoints. The second form will set breakpoints at each of the specified address up to a maximum of 10 breakpoints. The stack pointer in the register save area must point to a valid memory area in order for this command to work. If the program under test leaves interrupts enabled, execution may be interrupted by typing a C at the console. All registers are saved when interrupted and the program may be continued with a GO command.

5.9 Input I WORD

Inputs a byte of data from the specified port and displays it. A 16 bit port address is allowed.

5.10 Hex Arithmetic

H WORD WORD +,-,\,.*

This function performs the specified function on the two supplied parameters. The displayed result is always a 16 bit quantity.

5.11 Load Disk Sections L ADDRESS, Drive, Beginning Record #, # of Sectors

The specified number of sectors are loaded beginning at address. The first sector is specified by Beginning Record # . The records on a disk are numbered from 0, with track 0 sector 1 being record 0.

DRIVE #	DRIVE SELECT	DRIVE DESIGNATORS SWITCH 4 ^{OFF} ON	FORMAT EXPECTED
0	0		SS-DD 8 SPT
1	1		SS-DD 8 SPT
2	2		SS-DD 8 SPT
3	0		DS-DD 8 SPT
4	1		DS-DD 8 SPT

5	2	DS-DD 8 SPT
6	0	SS-DD 9 SPT
7	1	SS-DD 9 SPT
8	2	SS-DD 9 SPT
9	0	DS-DD 9 SPT
A	1	DS-DD 9 SPT
B	2	DS-DD 9 SPT

DRIVE DESIGNATORS SWITCH 4 ~~OFF~~ ^{ON} 8" Boot

0	0	SS-SD 8"
1	1	SS-SD 8"
2	0	SS-DD 8"
3	1	SS-DD 8"
4	0	DS-DD 8"
5	1	DS-DD 8"
6	2	5 1/4" SINGLE SIDED
7	2	5 1/4" DOUBLE SIDED

5.12 Move

M RANGE ADDRESS

MOVE moves a block of memory from the area specified by RANGE to the new area beginning at ADDRESS. Overlapping moves are allowed.

5.13 OUTPUT TO PORT

O WORD BYTE

Outputs the BYTE value to port specified in WORD.

5.14 REGISTER DISPLAY

R

R REGISTER NAME

Register - with no parameters, this command dumps the register save area.

Giving a register name as a parameter allows that register to be displayed and modified. The register name may be AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP, PC or F (upper case only): anything else will result in "BR Error". IP and PC both refer to the Instruction Pointer and F refers to the Flag register. For all except the Flag register, the current 16-bit value will be printed in hex, then a colon will appear as a prompt for the replacement value. Typing carriage return leaves the register unchanged; otherwise, type a WORD to replace.

The Flag register uses a system of two-letter mnemonics for each flag, as shown below:

FLAG	CLEAR	SET
Overflow	NV No Overflow	OV Overflow

Direction	UP Up (Incrementing)	DN Down (Decrementing)
Interrupt	DI Disabled Interrupts	EI Enabled Interrupts
Sign	PL Plus	NG Negative
Zero	NX Not Zero	ZR Zero
Auxiliary Carry	NA No Auxiliary Carry	AC Auxiliary Carry
Parity	PO Parity Odd	PE Parity Even
Carry	NC No Carry	CY Carry

Whenever the Flag register is displayed, all flags are displayed in this order. When the F register is specified with the R command, the flags are displayed and then the Monitor waits for any replacements to be made. Any number of two letter flag codes may be typed, and only those flags entered will be modified. If a flag has more than one code in the list, a "DF Error" (Double Flag) will result. If any code is not recognized, and "BF/Error" (Bad Flag) will occur. In either case, those flags up to the error have been changed, and those after the error have not.

After reset, all registers are set to zero except the segment registers are set to 40H, and the Stack Pointer, which is set to 0COOH. Flags are cleared except for interrupts. Execution on a Trace or Go command would thus begin at 400H, which is the first location after the interrupt table.

5.15 Search

S RANGE LIST

The specified range is searched for a byte or string of bytes specified in LIST. The address of each occurrence is listed at the console.

5.16 TRACE

T
T WORD

The number of instructions specified are traced. The register contents are printed after each step.

5.17 VERIFY

V TEST START COUNT ITERATIONS

TEST is a single digit number in the range 0 to 4 which specifies a particular memory test. START is a 4 digit hex number which specifies the starting paragraph number for testing. A paragraph number is the absolute memory address divided by four. For instance to begin testing memory at address 400H, specify 40 for

start. COUNT is the number of paragraphs to test. Each paragraph is 16 bytes. The minimum starting paragraph for testing should be 40H, as the memory below 40H is used by the monitor for working RAM. ITERATIONS is the number of times to test the specified range. An example command line to test all testable memory in a 128K byte system is as follows

*V 2 ~~40-1F00~~ 8

50 3F80 8

This command will test all but the first 1K of memory in a 128K system, which is used by the monitor and is tested at reset time, eight times. The tests available are as follows:

- TEST 0 - tests the specified range with all 0's
- TEST 1 - tests the specified range with all 1's
- TEST 2 - tests the specified range with an alternating 0,1's pattern and then its complement. In other words the test first fills memory with 55H and verifies it and then fill memory with AAH and verifies it.
- TEST 3 - tests memory with a shifting 1 pattern. The first iteration uses 1 in bit 0 and each succeeding iteration shifts it left one position. In order to test memory for all single bit combinations an iteration count of 8 should be specified.
- TEST 4 - tests the specified range with a shifting 0 pattern.

Bit 0 has a 0 and the remaining bits are set to 1's. Each iteration shifts the 0 one bit left. As in test 3 an iteration count of 8 should be specified in order to test all locations for each possible combination.

5.18 WRITE

W ADDRESS, Drive, Beginning Record #, # of Records

The specified number of sectors are written to the specified disk. First byte transferred is at address. See the drive designators under the LOAD command.



EPROM TYPE

prewrite compensation

RESET
SLAVE
CLR

AND SW1 2 3 4
ON 8 1/4"
OFF 5 1/4"
ON BOOT
OFF MONITOR

2. OVERVIEW OF THE 80186

2.1 The CPU

The 80186 CPU shares a common base architecture with the 8086, 8088 and 80286. It is completely object code compatible with the 8086/88. This architecture features four 16-bit general purpose registers (AX, BX, CX, DX) which may be used as operands in most arithmetic operations in either 8 or 16 bit units. It also features four 16-bit "pointer" registers (SI, DI, BP, SP) which may be used both in arithmetic operations and in accessing memory based variables. Four 16-bit segment registers (CS, DS, SS, ES) are provided allowing simple memory partitioning to aid construction of modular programs. Finally, it has a 16-bit instruction pointer and a 16-bit status register.

Physical memory addresses are generated by the 80186 identically to the 8086. The 16-bit segment value is left shifted 4 bits and then is added to an offset value which is derived from combinations of the pointer registers, the instruction pointer, and immediate values (see Figure 2). Any carry out of this addition is ignored. The result of this addition is a 20-bit physical address which is presented to the system memory.

The 80186 has a 16-bit ALU which performs 8 or 16-bit arithmetic and logical operations. It provides for data movement among registers, memory and I/O space. In addition, the CPU allows for high speed data transfer from one area of memory to another using string move instructions, and to or from an I/O port and memory using block I/O instructions. Finally, the CPU provides a

wealth of conditional branch and other control instructions.

In the 80186, as in the 8086, instruction fetching and instruction execution are performed by separate units: the bus interface unit and the execution unit, respectively. The 80186 also has a 6-byte prefetch queue as does the 8086. The 80188 has a 4-byte prefetch queue as does the 8088. As a program is executing, opcodes are fetched from memory by the bus interface unit and placed in this queue. Whenever the execution unit requires another instruction, it takes it out of the queue. Effective processor throughput is increased by adding this queue, since the bus interface unit may continue to fetch instructions while the execution unit executes a long instruction. Then, when the CPU completes this instruction, it does not have to wait for another instruction to be fetched from memory.

2.2 80186 CPU Enhancements

Although the 80186 is completely object code compatible with the 8086, most of the 8086 instructions require fewer clock cycles to execute on the 80186 than on the 8086 because of hardware enhancements in the bus interface unit and the execution unit. In addition, the 80186 provides many new instructions which simplify assembly language programming, enhance the performance of high level language implementations, and reduce object code sizes for the 80186. These new instructions are also included in the 80286. A complete description of the architecture and instruction execution of the 80186 can be found in volume I of the iAPX86/186 users manual. The algorithms for the new instructions are also given in appendix H of this note.

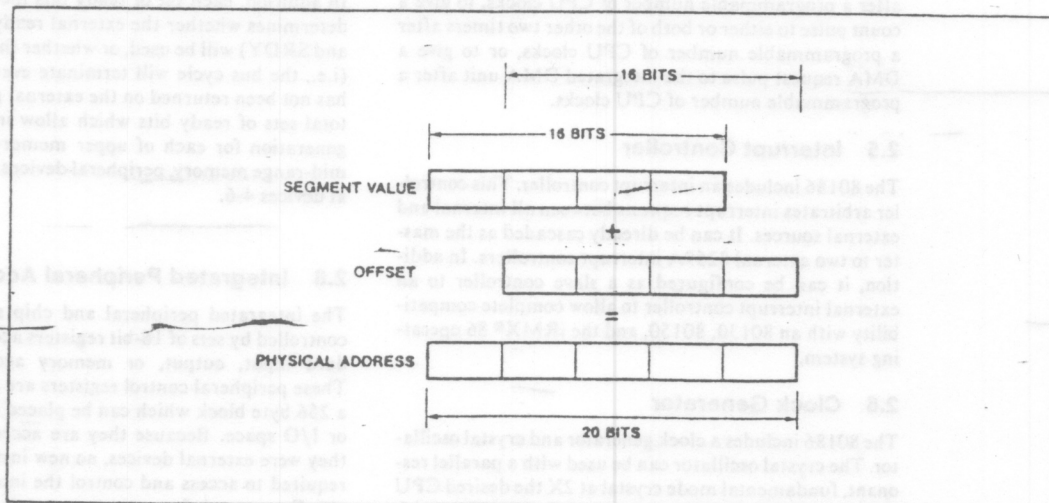


Figure 2. Physical Address Generation in the 80186

2.3 DMA Unit

The 80186 includes a DMA unit which provides two high speed DMA channels. This DMA unit will perform transfers to or from any combination of I/O space and memory space in either byte or word units. Every DMA cycle requires two to four bus cycles, one or two to fetch the data to an internal register, and one or two to deposit the data. This allows word data to be located on odd boundaries, or byte data to be moved from odd locations to even locations. This is normally difficult, since odd data bytes are transferred on the upper 8 data bits of the 16-bit data bus, while even data bytes are transferred on the lower 8 data bits of the data bus.

Each DMA channel maintains independent 20-bit source and destination pointers which are used to access the source and destination of the data transferred. Each of these pointers may independently address either I/O or memory space. After each DMA cycle, the pointers may be independently incremented, decremented, or maintained constant. Each DMA channel also maintains a transfer count which may be used to terminate a series of DMA transfers after a pre-programmed number of transfers.

2.4 Timers

The 80186 includes a timer unit which contains 3 independent 16-bit timer/counters. Two of these timers can be used to count external events, to provide waveforms derived from either the CPU clock or an external clock of any duty cycle, or to interrupt the CPU after a specified number of timer "events." The third timer counts only CPU clocks and can be used to interrupt the CPU after a programmable number of CPU clocks, to give a count pulse to either or both of the other two timers after a programmable number of CPU clocks, or to give a DMA request pulse to the integrated DMA unit after a programmable number of CPU clocks.

2.5 Interrupt Controller

The 80186 includes an interrupt controller. This controller arbitrates interrupt requests between all internal and external sources. It can be directly cascaded as the master to two external 8259A interrupt controllers. In addition, it can be configured as a slave controller to an external interrupt controller to allow complete compatibility with an 80130, 80150, and the iRMX[®] 86 operating system.

2.6 Clock Generator

The 80186 includes a clock generator and crystal oscillator. The crystal oscillator can be used with a parallel resonant, fundamental mode crystal at 2X the desired CPU clock speed (i.e., 16 MHz for an 8 MHz 80186), or with an external oscillator also at 2X the CPU clock. The output of the oscillator is internally divided by two to provide the 50% duty cycle CPU clock from which all

80186 system timing derives. The CPU clock is externally available, and all timing parameters are referenced to this externally available signal. The clock generator also provides ready synchronization for the processor.

2.7 Chip Select and Ready Generation Unit

The 80186 includes integrated chip select logic which can be used to enable memory or peripheral devices. Six output lines are used for memory addressing and seven output lines are used for peripheral addressing.

The memory chip select lines are split into 3 groups for separately addressing the major memory areas in a typical 8086 system: upper memory for reset ROM, lower memory for interrupt vectors, and mid-range memory for program memory. The size of each of these regions is user programmable. The starting location and ending location of lower memory and upper memory are fixed at 00000H and FFFFFH respectively; the starting location of the mid-range memory is user programmable.

Each of the seven peripheral select lines address one of seven contiguous 128 byte blocks above a programmable base address. This base address can be located in either memory or I/O space in order that peripheral devices may be I/O or memory mapped.

Each of the programmed chip select areas has associated with it a set of programmable ready bits. These ready bits control an integrated wait state generator. This allows a programmable number of wait states (0 to 3) to be automatically inserted whenever an access is made to the area of memory associated with the chip select area. In addition, each set of ready bits includes a bit which determines whether the external ready signals (ARDY and SRDY) will be used, or whether they will be ignored (i.e., the bus cycle will terminate even though a ready has not been returned on the external pins). There are 5 total sets of ready bits which allow independent ready generation for each of upper memory, lower memory, mid-range memory, peripheral devices 0-3 and peripheral devices 4-6.

2.8 Integrated Peripheral Accessing

The integrated peripheral and chip select circuitry is controlled by sets of 16-bit registers accessed using standard input, output, or memory access instructions. These peripheral control registers are all located within a 256 byte block which can be placed in either memory or I/O space. Because they are accessed exactly as if they were external devices, no new instruction types are required to access and control the integrated peripherals. For more information concerning the interfacing and accessing of the integrated 80186 peripherals not included in this note, please consult the 80186 data sheet, or volume II of the iAPX86/186 users manual.

3. USING THE 80186

3.1 Bus Interfacing to the 80186

3.1.1 OVERVIEW

The 80186 bus structure is very similar to the 8086 bus structure. It includes a multiplexed address/data bus, along with various control and status lines (see Table 1). Each bus cycle requires a minimum of 4 CPU clock cycles along with any number of wait states required to accommodate the speed access limitations of external memory or peripheral devices. The bus cycles initiated by the 80186 CPU are identical to the bus cycles initiated by the 80186 integrated DMA unit.

In the following discussion, all timing values given are for an 8 MHz 80186. Future speed selections of the part may have different values for the various parameters.

Each clock cycle of the 80186 bus cycle is called a "T" state, and are numbered sequentially T_1 , T_2 , T_3 , T_4 , and T_5 . Additional idle T states (T_i) can occur between T_4 and T_1 when the processor requires no bus activity (instruction fetches, memory writes, I/O reads, etc.). The ready signals control the number of wait states (T_w) inserted in each bus cycle. This number can vary from 0 to positive infinity.

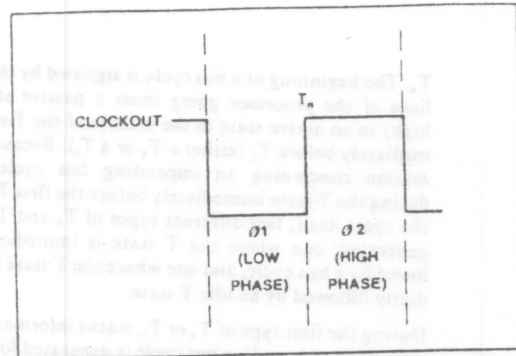


Figure 3. T-state in the 80186

The beginning of a T state is signaled by a high to low transition of the CPU clock. Each T state is divided into two phases, phase 1 (or the low phase) and phase 2 (or the high phase) which occur during the low and high levels of the CPU clock respectively (see Figure 3).

Different types of bus activity occur for all of the T-states (see Figure 4). Address generation information occurs during T_1 , data generation during T_2 , T_3 , T_4 , and T_5 .

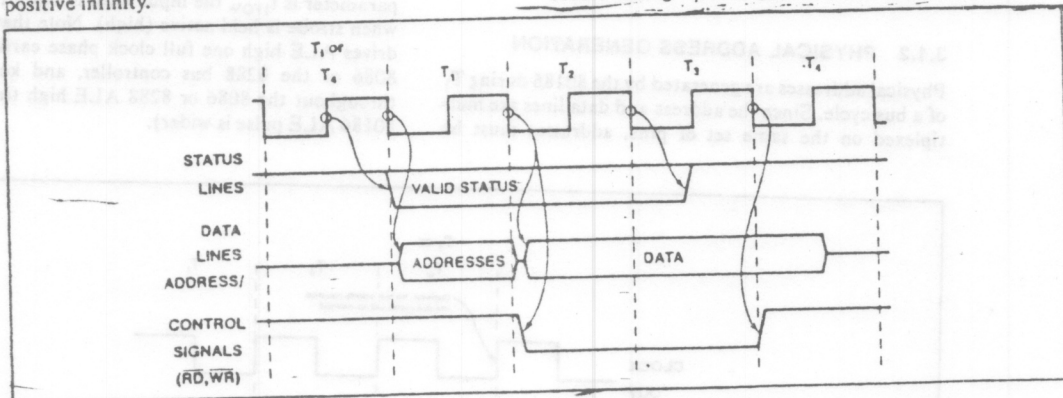


Figure 4. Example Bus Cycle of the 80186

Table 1. 80186 Bus Signals

Function	Signal Name
address/data	AD0-AD15
address/status	A16/S3-A19-S6, $\overline{\text{BHE}}/\text{S7}$
co-processor control	$\overline{\text{TEST}}$
local bus arbitration	HOLD, HLDA
local bus control	$\text{ALE}, \text{RD}, \text{WR}, \text{DT}/\text{R}, \text{DEN}$
multi-master bus	$\overline{\text{LOCK}}$
ready (wait) interface	SRDY, ARDY
status information	S0-S2

T_4 . The beginning of a bus cycle is signaled by the status lines of the processor going from a passive state (all high) to an active state in the middle of the T-state immediately before T_1 (either a T_4 or a T_1). Because information concerning an impending bus cycle occurs during the T-state immediately before the first T-state of the cycle itself, two different types of T_4 and T_1 can be generated: one where the T state is immediately followed by a bus cycle, and one where the T state is immediately followed by an idle T state.

During the first type of T_4 or T_1 , status information concerning the impending bus cycle is generated for the bus cycle immediately to follow. This information will be available no later than t_{CHSV} (55ns) after the low-to-high transition of the 80186 clock in the middle of the T state. During the second type of T_4 or T_1 the status outputs remain inactive (high), since no bus cycle is to be started. This means that the decision per the nature of a T_4 or T_1 state (i.e., whether it is immediately followed by a T_1 or a T_4) is decided at the beginning of the T-state immediately preceding the T_4 or T_1 (see Figure 5). This has consequences for the bus latency time (see section 3.3.2 on bus latency).

3.1.2 PHYSICAL ADDRESS GENERATION

Physical addresses are generated by the 80186 during T_1 of a bus cycle. Since the address and data lines are multiplexed on the same set of pins, addresses must be

latched during T_1 if they are required to remain stable for the duration of the bus cycle. To facilitate latching of the physical address, the 80186 generates an active high ALE (Address Latch Enable) signal which can be directly connected to a transparent latch's strobe input.

Figure 6 illustrates the physical address generation parameters of the 80186. Addresses are guaranteed valid no greater than t_{CLAV} (44ns) after the beginning of T_1 , and remain valid at least t_{CLAX} (10ns) after the end of T_1 . The ALE signal is driven high in the middle of the T state (either T_4 or T_1) immediately preceding T_1 and is driven low in the middle of T_1 , no sooner than t_{AVAL} (30 ns) after addresses become valid. This parameter (t_{AVAL}) is required to satisfy the address latch set-up times of address valid until strobe inactive. Addresses remain stable on the address/data bus at least t_{LLAX} (30 ns) after ALE goes inactive to satisfy address latch hold times of strobe inactive to address invalid.

Because ALE goes high long before addresses become valid, the delay through the address latches will be chiefly the propagation delay through the latch rather than the delay from the latch strobe, which is typically longer than the propagation delay. For the Intel 8282 latch, this parameter is t_{VOW} the input valid to output valid delay when strobe is held active (high). Note that the 80186 drives ALE high one full clock phase earlier than the 8086 or the 8288 bus controller, and keeps it high throughout the 8086 or 8288 bus high time (i.e., the 80186 ALE pulse is wider).

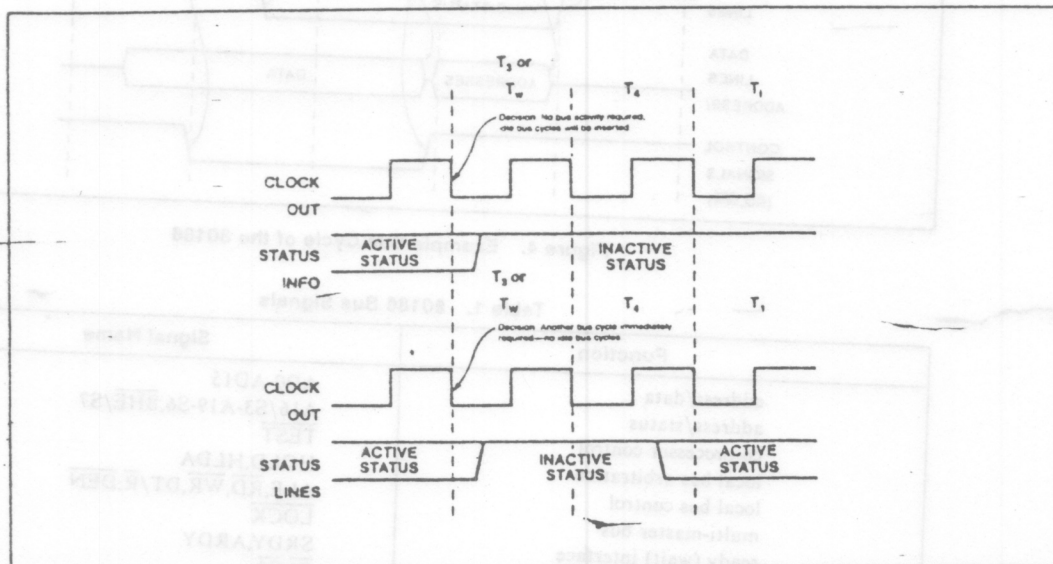


Figure 5. Active-Inactive Status Transitions in the 80186

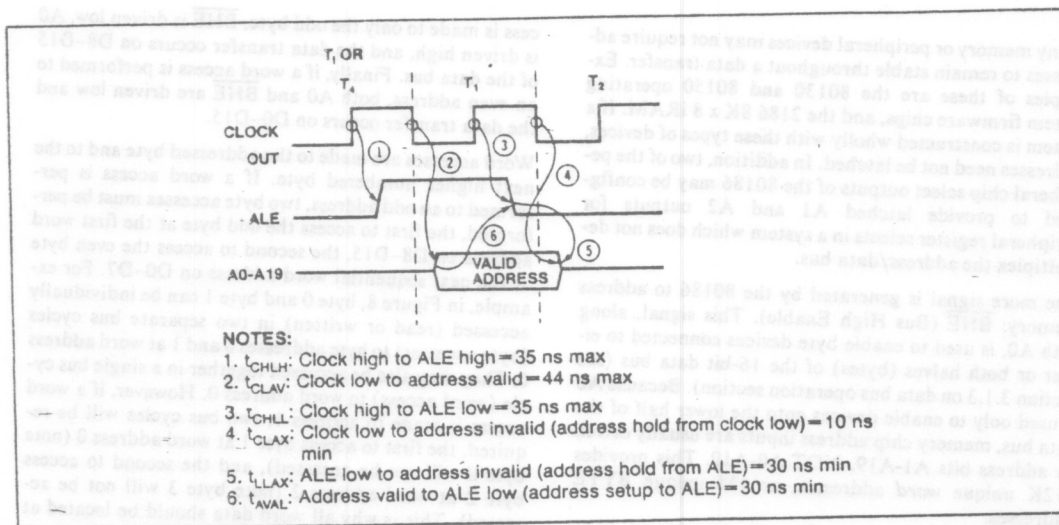


Figure 6. Address Generation Timing of the 80186

A typical circuit for latching physical addresses is shown in Figure 7. This circuit uses 3 8282 transparent octal non-inverting latches to demultiplex all 20 address bits provided by the 80186. Typically, the upper 4 address bits are used only to select among various memory components or subsystems, so when the integrated chip se-

lects (see section 8) are used, these upper bits need not be latched. The worst case address generation time from the beginning of T_1 (including address latch propagation time (t_{IVOV}) of the Intel 8282) for the circuit is:

$$t_{CLAV} (44\text{ns}) + t_{IVOV} (30\text{ns}) = 74\text{ns}$$

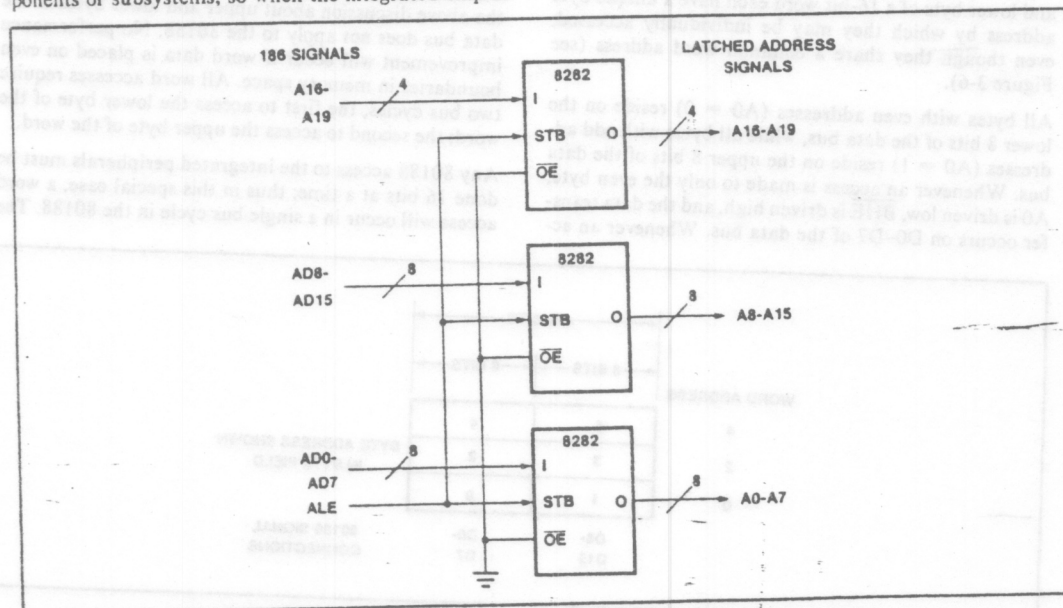


Figure 7. Demultiplexing the Address Bus of the 80186

Many memory or peripheral devices may not require addresses to remain stable throughout a data transfer. Examples of these are the 80130 and 80150 operating system firmware chips, and the 2186 8K x 8 iRAM. If a system is constructed wholly with these types of devices, addresses need not be latched. In addition, two of the peripheral chip select outputs of the 80186 may be configured to provide latched A1 and A2 outputs for peripheral register selects in a system which does not demultiplex the address/data bus.

One more signal is generated by the 80186 to address memory: $\overline{\text{BHE}}$ (Bus High Enable). This signal, along with A0, is used to enable byte devices connected to either or both halves (bytes) of the 16-bit data bus (see section 3.1.3 on data bus operation section). Because A0 is used only to enable devices onto the lower half of the data bus, memory chip address inputs are usually driven by address bits A1-A19, NOT A0-A19. This provides 512K unique word addresses, or 1M unique BYTE addresses.

Of course, $\overline{\text{BHE}}$ is not present on the 8 bit 80188. All data transfers occur on the 8 bits of the data bus.

3.1.3 80186 DATA BUS OPERATION

Throughout T_2 , T_3 , T_4 , and T_5 of a bus cycle the multiplexed address/data bus becomes a 16-bit data bus. Data transfers on this bus may be either in bytes or in words. All memory is byte addressable, that is, the upper and lower byte of a 16-bit word each have a unique byte address by which they may be individually accessed, even though they share a common word address (see Figure 3-6).

All bytes with even addresses ($A_0 = 0$) reside on the lower 8 bits of the data bus, while all bytes with odd addresses ($A_0 = 1$) reside on the upper 8 bits of the data bus. Whenever an access is made to only the even byte, A0 is driven low, $\overline{\text{BHE}}$ is driven high, and the data transfer occurs on D0-D7 of the data bus. Whenever an ac-

cess is made to only the odd byte, $\overline{\text{BHE}}$ is driven low, A0 is driven high, and the data transfer occurs on D8-D15 of the data bus. Finally, if a word access is performed to an even address, both A0 and $\overline{\text{BHE}}$ are driven low and the data transfer occurs on D0-D15.

Word accesses are made to the addressed byte and to the next higher numbered byte. If a word access is performed to an odd address, two byte accesses must be performed, the first to access the odd byte at the first word address on D8-D15, the second to access the even byte at the next sequential word address on D0-D7. For example, in Figure 8, byte 0 and byte 1 can be individually accessed (read or written) in two separate bus cycles (byte accesses) to byte addresses 0 and 1 at word address 0. They may also be accessed together in a single bus cycle (word access) to word address 0. However, if a word access is made to address 1, two bus cycles will be required, the first to access byte 1 at word address 0 (note byte 0 will not be accessed), and the second to access byte 2 at word address 2 (note byte 3 will not be accessed). This is why all word data should be located at even addresses to maximize processor performance.

When byte reads are made, the data returned on the half of the data bus not being accessed is ignored. When byte writes are made, the data driven on the half of the data bus not being written is indeterminate.

3.1.4 80188 DATA BUS OPERATION

Because the 80188 externally has only an 8 bit data bus, the above discussion about upper and lower bytes of the data bus does not apply to the 80188. No performance improvement will occur if word data is placed on even boundaries in memory space. All word accesses require two bus cycles, the first to access the lower byte of the word; the second to access the upper byte of the word.

Any 80188 access to the integrated peripherals must be done 16 bits at a time; thus in this special case, a word access will occur in a single bus cycle in the 80188. The

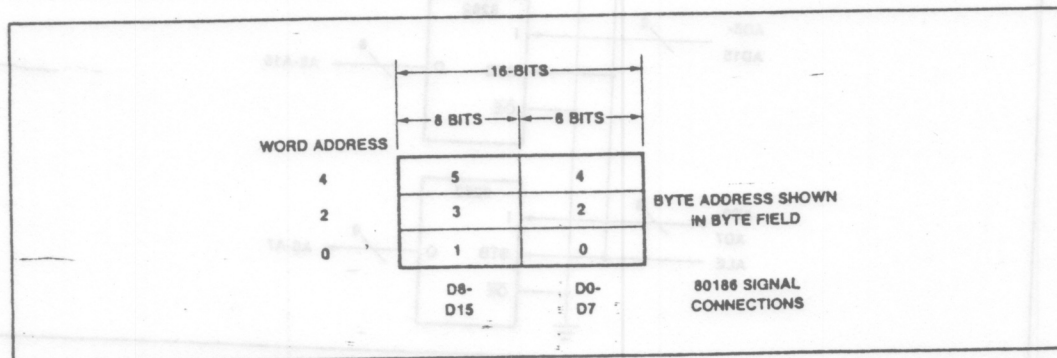


Figure 8. Physical Memory Byte/Word Addressing in the 80186

external data bus will record only a single byte being transferred, however.

3.1.5 GENERAL DATA BUS OPERATION

Because of the bus drive capabilities of the 80186 (200pF, sinking 2mA, sourcing 400uA, roughly twice that of the 8086), this bus may not require additional buffering in many small systems. If data buffers are not used in the system, care should be taken not to allow bus contention between the 80186 and the devices directly connected to the 80186 data bus. Since the 80186 floats the address/data bus before activating any command lines, the only requirement on a directly connected device is that it floats its output drivers after a read *BEFORE* the 80186 begins to drive address information for the next bus cycle. The parameter of interest here is the minimum time from $\overline{RD}$ inactive until addresses active for the next bus cycle (t_{RHAV}) which has a minimum value of 85ns. If the memory or peripheral device cannot disable its output drivers in this time, data buffers will be required to prevent both the 80186 and the peripheral or memory device from driving these lines concurrently. Note, this parameter is unaffected by the addition of wait states. Data buffers solve this problem because their output float times are typically much faster than the 80186 required minimum.

If buffers are required, the 80186 provides a $\overline{DEN}$ (Data Enable) and $DT/\overline{R}$ (Data Transmit/Receive) signals to simplify buffer interfacing. The $\overline{DEN}$ and $DT/\overline{R}$ sig-

nals are activated during all bus cycles, whether or not the cycle addresses buffered devices. The $\overline{DEN}$ signal is driven low whenever the processor is either ready to receive data (during a read) or when the processor is ready to send data (during a write) (that is, any time during an active bus cycle when address information is not being generated on the address/data pins). In most systems, the $\overline{DEN}$ signal should NOT be directly connected to the $\overline{OE}$ input of buffers, since unbuffered devices (or other buffers) may be directly connected to the processor's address/data pins. If $\overline{DEN}$ were directly connected to several buffers, contention would occur during read cycles, as many devices attempt to drive the processor bus. Rather, it should be a factor (along with the chip selects for buffered devices) in generating the output enable input of a bi-directional buffer.

The $DT/\overline{R}$ signal determines the direction of data propagation through the bi-directional bus buffers. It is high whenever data is being driven out from the processor, and is low whenever data is being read into the processor. Unlike the $\overline{DEN}$ signal, it may be directly connected to bus buffers, since this signal does not usually directly enable the output drivers of the buffer. An example data bus subsystem supporting both buffered and unbuffered devices is shown in Figure 9. Note that the A side of the 8286 buffer is connected to the 80186, the B side to the external device. The B side of the buffer has greater drive capacity than the A side (since it is meant to drive much greater loads). The $DT/\overline{R}$ signal can directly drive the T (transmit) signal of the buffer, since it has the correct polarity for this configuration.

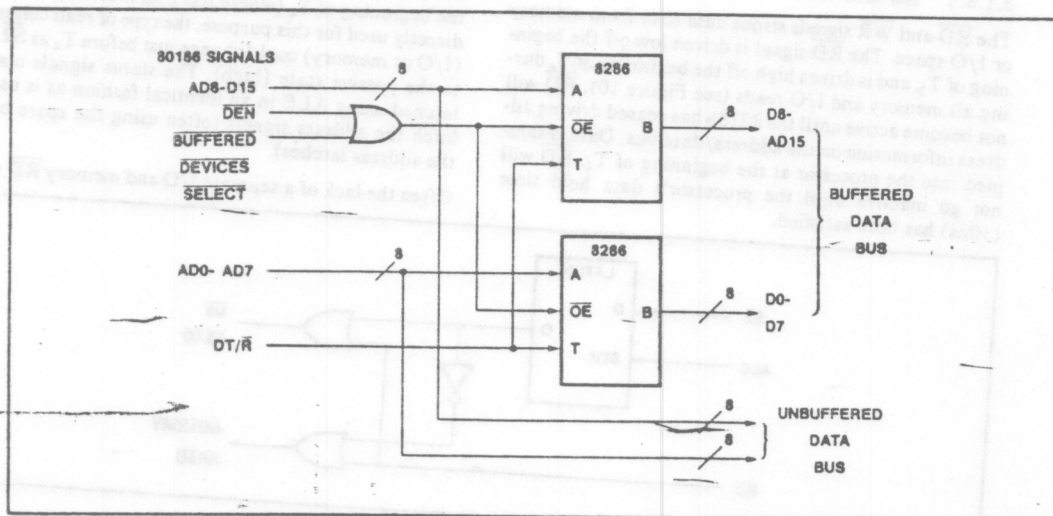


Figure 9. Example 80186 Buffered/Unbuffered Data Bus

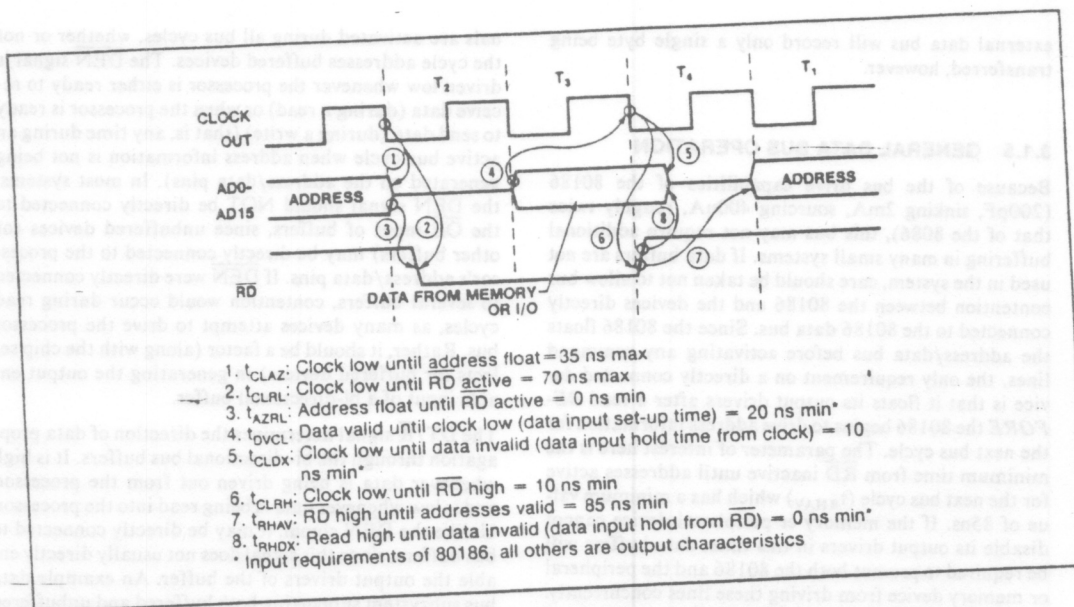


Figure 10. Read Cycle Timing of the 80186

3.1.6 CONTROL SIGNALS

The 80186 directly provides the control signals $\overline{RD}$, $\overline{WR}$, $\overline{LOCK}$ and $\overline{TEST}$. In addition, the 80186 provides the status signals $\overline{S0}$ - $\overline{S2}$ and $\overline{S6}$ from which all other required bus control signals can be generated.

3.1.6.1 $\overline{RD}$ and $\overline{WR}$

The $\overline{RD}$ and $\overline{WR}$ signals strobe data to or from memory or I/O space. The $\overline{RD}$ signal is driven low off the beginning of T_2 , and is driven high off the beginning of T_4 during all memory and I/O reads (see Figure 10). $\overline{RD}$ will not become active until the 80186 has ceased driving address information on the address/data bus. Data is sampled into the processor at the beginning of T_4 . $\overline{RD}$ will not go inactive until the processor's data hold time (10ns) has been satisfied.

Note that the 80186 does not provide separate I/O and memory $\overline{RD}$ signals. If separate I/O read and memory read signals are required, they can be synthesized using the $\overline{S2}$ signal (which is low for all I/O operations and high for all memory operations) and the $\overline{RD}$ signal (see Figure 11). It should be noted that if this approach is used, the $\overline{S2}$ signal will require latching, since the $\overline{S2}$ signal (like $\overline{S0}$ and $\overline{S1}$) goes to a passive state well before the beginning of T_4 (where $\overline{RD}$ goes inactive). If $\overline{S2}$ was directly used for this purpose, the type of read command (I/O or memory) could change just before T_4 as $\overline{S2}$ goes to the passive state (high). The status signals may be latched using ALE in an identical fashion as is used to latch the address signals (often using the spare bits in the address latches).

Often the lack of a separate I/O and memory $\overline{RD}$ signal

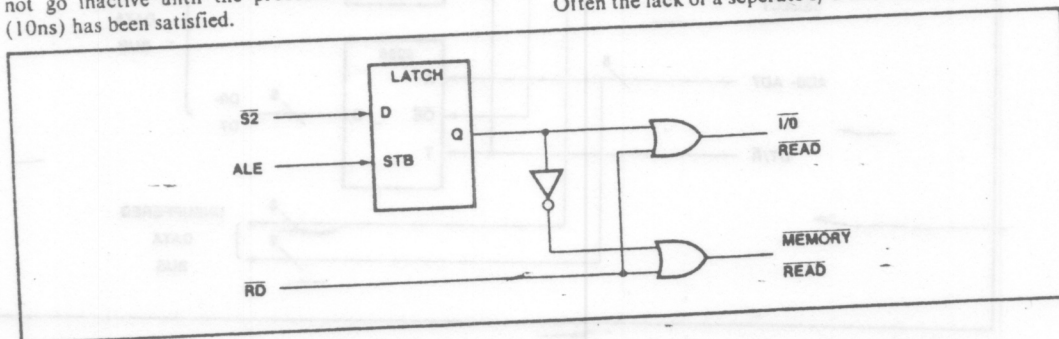


Figure 11. Generating I/O and Memory Read Signals from the 80186.

is not important in an 80186 system. Each of the 80186 chip select signals will respond on only one of memory or I/O accesses (the memory chip selects respond only to accesses memory space; the peripheral chip selects can respond to accesses in either I/O or memory space, at programmer option). Thus, the chip select signal enables the external device only during accesses to the proper address in the proper space.

The $\overline{WR}$ signal is also driven low off the beginning of T_2 and driven high off the beginning of T_4 . Like the $\overline{RD}$ signal, the $\overline{WR}$ signal is active for all memory and I/O writes, and also like the $\overline{RD}$ signal, separate I/O and memory writes may be generated using the latched S_2 signal along with the $\overline{WR}$ signal (see Figure 12). More

importantly, however, is the active going edge of write. At the time $\overline{WR}$ makes its active (high to low) transition, valid write data is NOT present on the data bus. This has consequences when using this signal as a write enable signal for DRAMs and iRAMs since both of these devices require that the write data be stable on the data bus at the time of the inactive to active transition of the $\overline{WE}$ signal. In DRAM applications, this problem is solved by a DRAM controller (such as the Intel 8207 or 8203), while with iRAMs this problem may be solved by placing cross-coupled NAND gates between the CPU and the iRAMs on the $\overline{WR}$ line (see Figure 13). This will delay the active going edge of the $\overline{WR}$ signal to the iRAMs by a clock phase, allowing valid data to be driven onto the data bus.

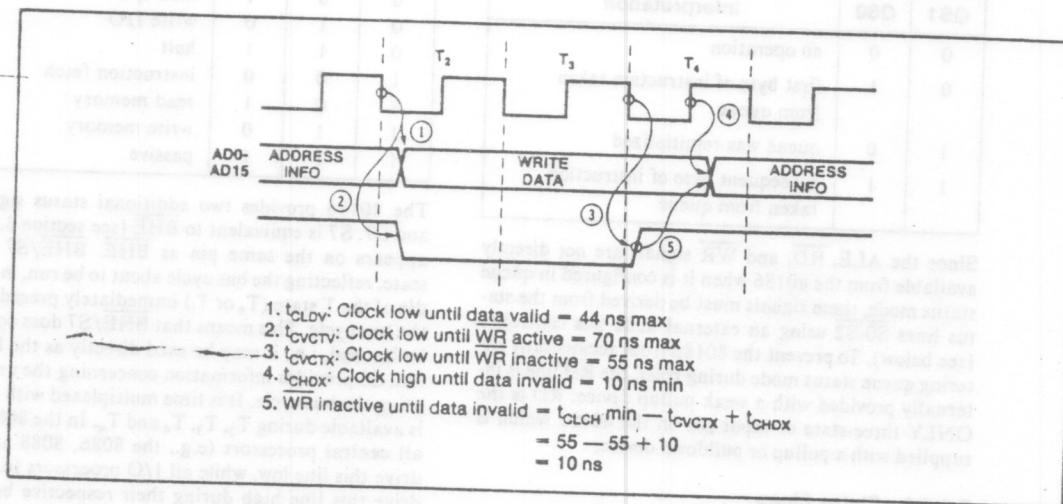


Figure 12. Write Cycle Timing of the 80186

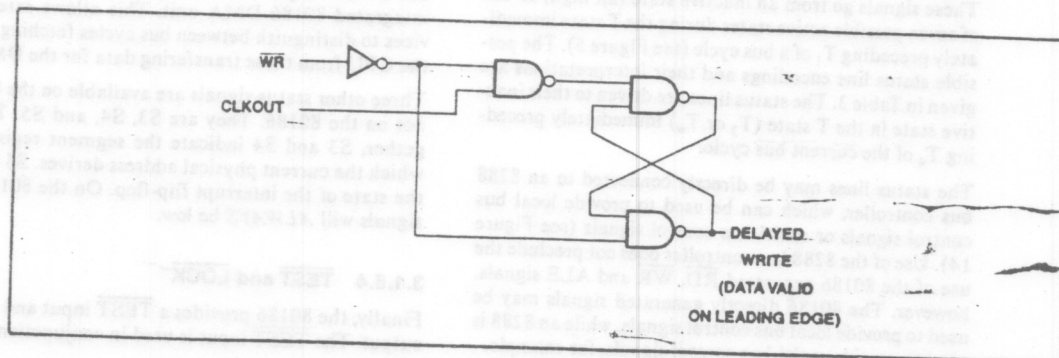


Figure 13. Synthesizing Delayed Write from the 80186

3.1.6.2 Queue Status Signals

If the $\overline{RD}$ line is externally grounded during reset and remains grounded during processor operation, the 80186 will enter "queue status" mode. When in this mode, the $\overline{WR}$ and ALE signals become queue status outputs, reflecting the status of the internal prefetch queue during each clock cycle. These signals are provided to allow a processor extension (such as the Intel 8087 floating point processor) to track execution of instructions within the 80186. The interpretation of QS0 (ALE) and QS1 ($\overline{WR}$) are given in Table 2. These signals change on the high-to-low clock transition, one clock phase earlier than on the 8086. Note that since execution unit operation is independent of bus interface unit operation, queue status lines may change in any T state.

Table 2. 80186 Queue Status

QS1	QS0	Interpretation
0	0	no operation
0	1	first byte of instruction taken from queue
1	0	queue was reinitialized
1	1	subsequent byte of instruction taken from queue

Since the ALE, $\overline{RD}$, and $\overline{WR}$ signals are not directly available from the 80186 when it is configured in queue status mode, these signals must be derived from the status lines S0-S2 using an external 8288 bus controller (see below). To prevent the 80186 from accidentally entering queue status mode during reset, the $\overline{RD}$ line is internally provided with a weak pullup device. $\overline{RD}$ is the ONLY three-state or input pin on the 80186 which is supplied with a pullup or pulldown device.

3.1.6.3 Status Lines

The 80186 provides 3 status outputs which are used to indicate the type of bus cycle currently being executed. These signals go from an inactive state (all high) to one of seven possible active states during the T state immediately preceding T_1 of a bus cycle (see Figure 5). The possible status line encodings and their interpretations are given in Table 3. The status lines are driven to their inactive state in the T state (T_3 or T_w) immediately preceding T_4 of the current bus cycle.

The status lines may be directly connected to an 8288 bus controller, which can be used to provide local bus control signals or multi-bus control signals (see Figure 14). Use of the 8288 bus controller does not preclude the use of the 80186 generated $\overline{RD}$, $\overline{WR}$ and ALE signals, however. The 80186 directly generated signals may be used to provide local bus control signals, while an 8288 is used to provide multi-bus control signals, for example.

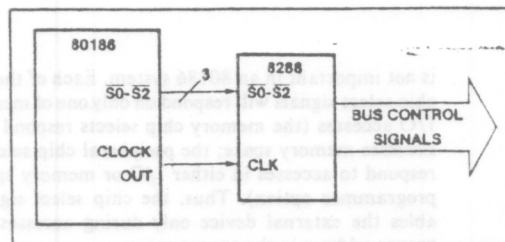


Figure 14. 80186/8288 Bus Controller Interconnection

Table 3. 80186 Status Line Interpretation

S2	S1	S0	Operation
0	0	0	interrupt acknowledge
0	0	1	read I/O
0	1	0	write I/O
0	1	1	halt
1	0	0	instruction fetch
1	0	1	read memory
1	1	0	write memory
1	1	1	passive

The 80186 provides two additional status signals: S6 and S7. S7 is equivalent to BHE (see section 3.1.2) and appears on the same pin as BHE. BHE/S7 changes state, reflecting the bus cycle about to be run, in the middle of the T state (T_4 or T_1) immediately preceding T_1 of the bus cycle. This means that BHE/S7 does not need to be latched, i.e., it may be used directly as the BHE signal. S6 provides information concerning the unit generating the bus cycle. It is time multiplexed with A19, and is available during T_2 , T_3 , T_4 and T_w . In the 8086 family, all central processors (e.g., the 8086, 8088 and 8087) drive this line low, while all I/O processors (e.g., 8089) drive this line high during their respective bus cycles. Following this scheme, the 80186 drives this line low whenever the bus cycle is generated by the 80186 CPU, but drives it high when the bus cycle is generated by the integrated 80186 DMA unit. This allows external devices to distinguish between bus cycles fetching data for the CPU from those transferring data for the DMA unit.

Three other status signals are available on the 8086 but not on the 80186. They are S3, S4, and S5. Taken together, S3 and S4 indicate the segment register from which the current physical address derives. S5 indicates the state of the interrupt flip-flop. On the 80186, these signals will ALWAYS be low.

3.1.6.4 TEST and LOCK

Finally, the 80186 provides a TEST input and a LOCK output. The TEST input is used in conjunction with the

processor WAIT instruction. It is typically driven by a processor extension (like the 8087) to indicate whether it is busy. Then, by executing the WAIT (or FWAIT) instruction, the central processor may be forced to temporarily suspend program execution until the processor extension indicates that it is idle by driving the TEST line low.

The $\overline{\text{LOCK}}$ output is driven low whenever the data cycles of a LOCKED instruction are executed. A LOCKED instruction is generated whenever the LOCK prefix occurs immediately before an instruction. The LOCK prefix is active for the single instruction immediately following the LOCK prefix. This signal is used to indicate to a bus arbiter (e.g., the 8289) that a series of locked data transfers is occurring. The bus arbiter should under no circumstances release the bus while locked transfers are occurring. The 80186 will not recognize a bus HOLD, nor will it allow DMA cycles to be run by the integrated DMA controller during locked data transfers. LOCKED transfers are used in multiprocessor systems to access memory based semaphore variables which control access to shared system resources (see AP-106, "Multiprogramming with the iAPX88 and iAPX86 Microsystems," by George Alexy (Sept. 1980)).

On the 80186, the $\overline{\text{LOCK}}$ signal will go active during T_1 of the first DATA cycle of the locked transfer. It is driven inactive 3 T-states after the beginning of the last DATA cycle of the locked transfer. On the 8086, the LOCK signal is activated immediately after the LOCK prefix is executed. The LOCK prefix may be executed well before the processor is prepared to perform the locked data transfer. This has the unfortunate consequence of activating the LOCK signal before the first LOCKED data cycle is performed. Since $\overline{\text{LOCK}}$ is active before the processor requires the bus for the data transfer, opcode pre-fetching can be LOCKED. However, since the 80186 does not activate the LOCK signal until the processor is ready to actually perform the locked transfer, locked pre-fetching will not occur with the 80186.

Note that the $\overline{\text{LOCK}}$ signal does not remain active until the end of the last data cycle of the locked transfer. This may cause problems in some systems if, for example, the processor requests memory access from a dual ported RAM array and is denied immediate access (because of a DRAM refresh cycle, for example). When the processor finally is able to gain access to the RAM array, it may have already dropped its $\overline{\text{LOCK}}$ signal, thus allowing the dual port controller to give the other port access to the RAM array instead. An example circuit which can be used to hold $\overline{\text{LOCK}}$ active until a RDY has been received by the 80186 is shown in Figure 15.

3.1.7 HALT TIMING

A HALT bus cycle is used to signal the world that the

80186 CPU has executed a HLT instruction. It differs from a normal bus cycle in two important ways.

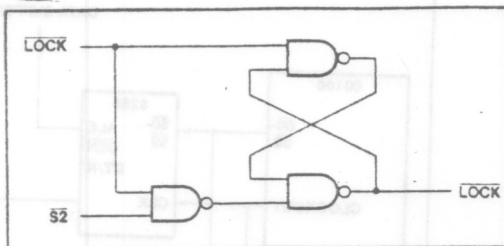


Figure 15. Circuit Holding $\overline{\text{LOCK}}$ Active Until Ready is Returned

The first way in which a HALT bus cycle differs from a normal bus cycle is that since the processor is entering a halted state, none of the control lines (RD or WR) will be driven active. Address and data information will not be driven by the processor, and no data will be returned. The second way a HALT bus cycle differs from a normal bus cycle is that the S_0 - S_2 status lines go to their passive state (all high) during T_2 of the bus cycle, well before they go to their passive state during a normal bus cycle.

Like a normal bus cycle, however, ALE is driven active. Since no valid address information is present, the information strobed into the address latches should be ignored. This ALE pulse can be used, however, to latch the HALT status from the S_0 - S_2 status lines.

The processor being halted does not interfere with the operation of any of the 80186 integrated peripheral units. This means that if a DMA transfer is pending while the processor is halted, the bus cycles associated with the DMA transfer will run: In fact, DMA latency time will improve while the processor is halted because the DMA unit will not be contending with the processor for access to the 80186 bus (see section 4.4.1).

3.1.8 8288 AND 8289 INTERFACING

The 8288 and 8289 are the bus controller and multi-master bus arbitration devices used with the 8086 and 8088. Because the 80186 bus is similar to the 8086 bus, they can be directly used with the 80186. Figure 16 shows an 80186 interconnection to these two devices.

The 8288 bus controller generates control signals ($\overline{\text{RD}}$, $\overline{\text{WR}}$, ALE, DT/R, $\overline{\text{DEN}}$, etc.) for an 8086 maximum mode system. It derives its information by decoding status lines S_0 - S_2 of the processor. Because the 80186 and the 8086 drive the same status information on these lines, the 80186 can be directly connected to the 8288 just as in an 8086 system. Using the 8288 with the 80186 does not prevent using the 80186 control signals directly. Many systems require both local bus control signals and system bus control signals. In this type of system, the 80186 lines could be used as the local signals, with the

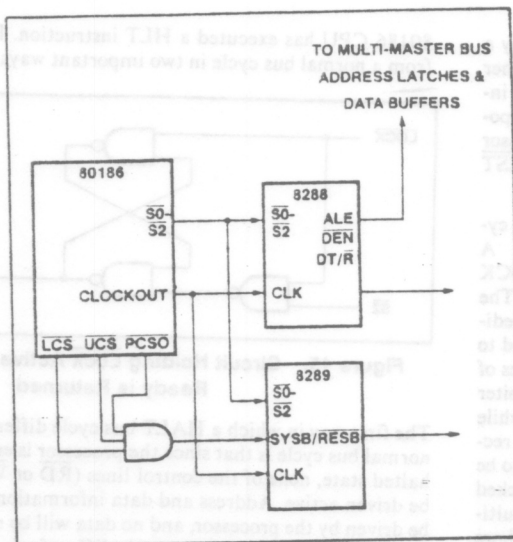


Figure 16. 80186/8288/8289 Interconnection

8288 lines used as the system signals. Note that in an 80186 system, the 8288 generated ALE pulse occurs later than that of the 80186 itself. In many multimaster bus systems, the 8288 ALE pulse should be used to strobe the addresses into the system bus address latches to insure that the address hold times are met.

The 8289 bus-arbiter arbitrates the use of a multi-master system bus among various devices each of which can become the bus master. This component also decodes status lines S0-S2 of the processor directly to determine when the system bus is required. When the system bus is required, the 8289 forces the processor to wait until it

has acquired control of the bus, then it allows the processor to drive address, data and control information onto the system bus. The system determines when it requires system bus resources by an address decode. Whenever the address being driven coincides with the address of an on-board resource, the system bus is not required and thus will not be requested. The circuit shown factors the 80186 chip select lines to determine when the system bus should be requested, or when the 80186 request can be satisfied using a local resource.

3.1.9 READY INTERFACING

The 80186 provides two ready lines, a synchronous ready (SRDY) line and an asynchronous ready (ARDY) line. These lines signal the processor to insert wait states (T_w) into a CPU bus cycle. This allows slower devices to respond to CPU service requests (reads or writes). Wait states will only be inserted when both ARDY and SRDY are low, i.e., only one of ARDY or SRDY need be active to terminate a bus cycle. Any number of wait states may be inserted into a bus cycle. The 80186 will ignore the RDY inputs during any accesses to the integrated peripheral registers, and to any area where the chip select ready bits indicate that the external ready should be ignored.

The timing required by the two RDY lines is different. The ARDY line is meant to be used with asynchronous ready inputs. Thus, inputs to this line will be internally synchronized to the CPU clock before being presented to the processor. The synchronization circuitry used with the ARDY line is shown in Figure 17. Figure 18A and 18B show valid and invalid transitions of the ARDY line (and subsequent wait state insertion). The first flip-flop is used to "resolve" the asynchronous transition of the ARDY line. It will achieve a definite level (either high or low) before its output is latched into the second flip-

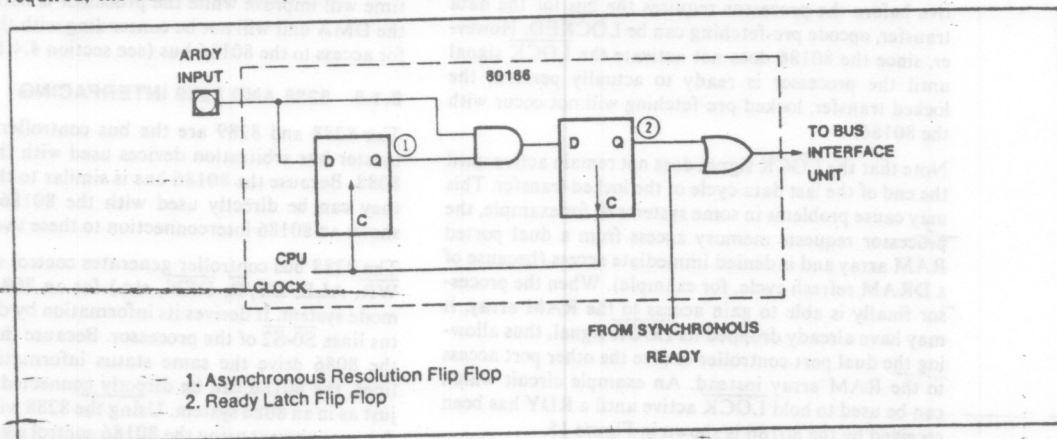


Figure 17. Asynchronous Ready Circuitry of the 80186

flop for presentation to the CPU. When latched high, it allows the level present on the ARDY line to pass directly to the CPU; when latched low, it forces not ready to be presented to the CPU (see Appendix B for 80186 synchronizer information).

With this scheme, notice that only the active going edge of the ARDY signal is synchronized. Once the synchronization flip-flop has sampled high, the ARDY input directly drives the RDY flip-flop. Since inputs to this RDY flip-flop must satisfy certain setup and hold times, it is important that these setup and hold times ($t_{\text{ARYLCL}} = 35\text{ ns}$ and $t_{\text{CHARYX}} = 15\text{ ns}$ respectively) be satisfied

by any inactive going transition of the ARDY line. The reason ARDY is implemented in this manner is to allow a slow device the greatest amount of time to respond with a not ready after it has been selected. In a normally ready system, a slow device must respond with a not ready quickly after it has been selected to prevent the processor from continuing and accessing invalid data from the slow device. By implementing ARDY in the above fashion, the slow device has an additional clock phase to respond with a not ready.

If RDY is sampled active into the RDY flip-flop at the beginning of T_3 or T_w (meaning that ARDY was sam-

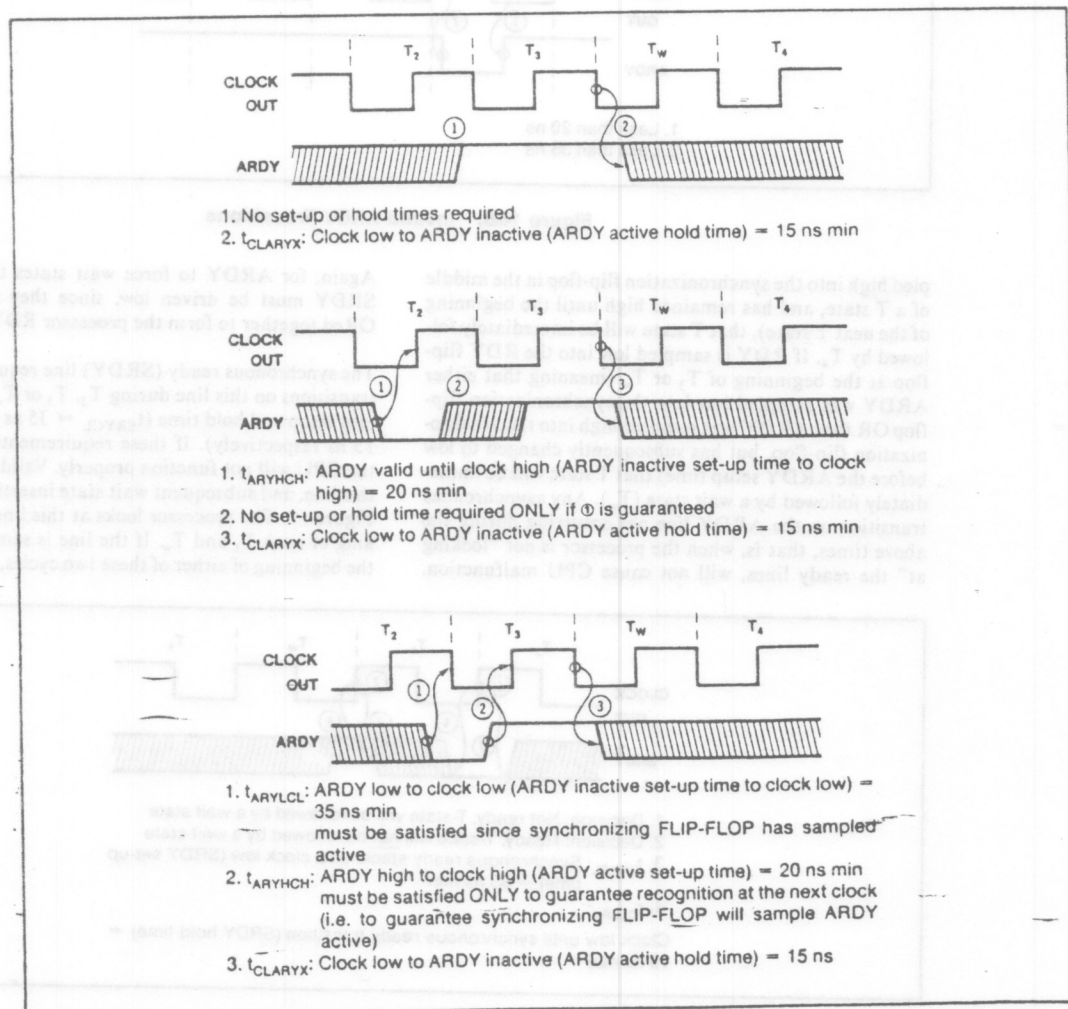


Figure 18A. Valid ARDY Transitions

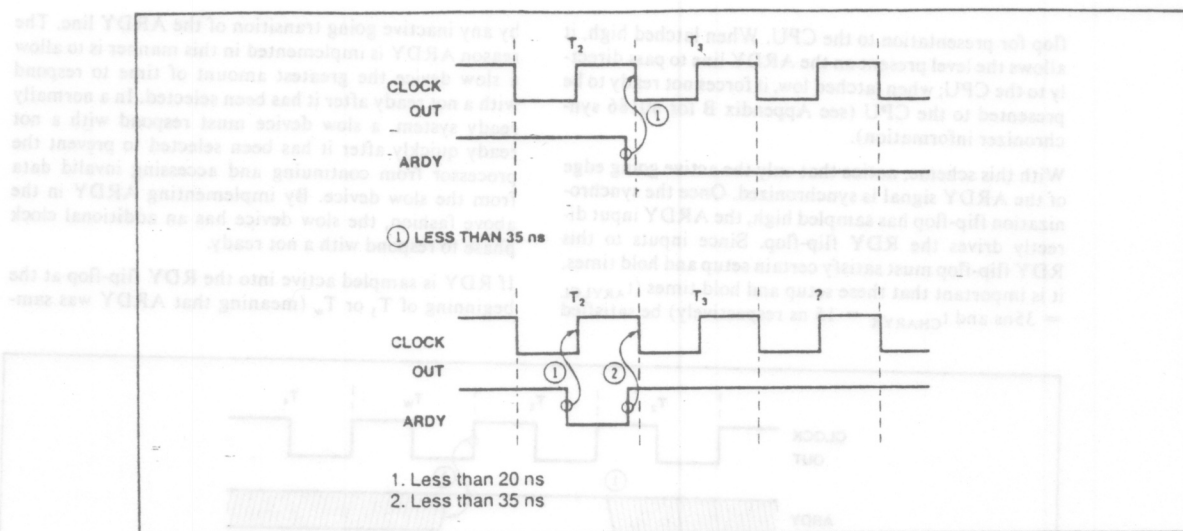


Figure 18B. Invalid ARDY Transitions

pled high into the synchronization flip-flop in the middle of a T state, and has remained high until the beginning of the next T state), that T state will be immediately followed by T_4 . If RDY is sampled low into the RDY flip-flop at the beginning of T_3 or T_w (meaning that either ARDY was sampled low into the synchronization flip-flop OR that ARDY was sampled high into the synchronization flip-flop, but has subsequently changed to low before the ARDY setup time) that T state will be immediately followed by a wait state (T_w). Any asynchronous transition on the ARDY line not occurring during the above times, that is, when the processor is not "looking at" the ready lines, will not cause CPU malfunction.

Again, for ARDY to force wait states to be inserted, SRDY must be driven low, since they are internally ORed together to form the processor RDY signal.

The synchronous ready (SRDY) line requires that ALL transitions on this line during T_2 , T_3 or T_w satisfy a certain setup and hold time ($t_{SRVCL} = 35$ ns and $t_{CLSRV} = 15$ ns respectively). If these requirements are not met, the CPU will not function properly. Valid transitions on this line, and subsequent wait state insertion is shown in Figure 19. The processor looks at this line at the beginning of each T_3 and T_w . If the line is sampled active at the beginning of either of these two cycles, that cycle will

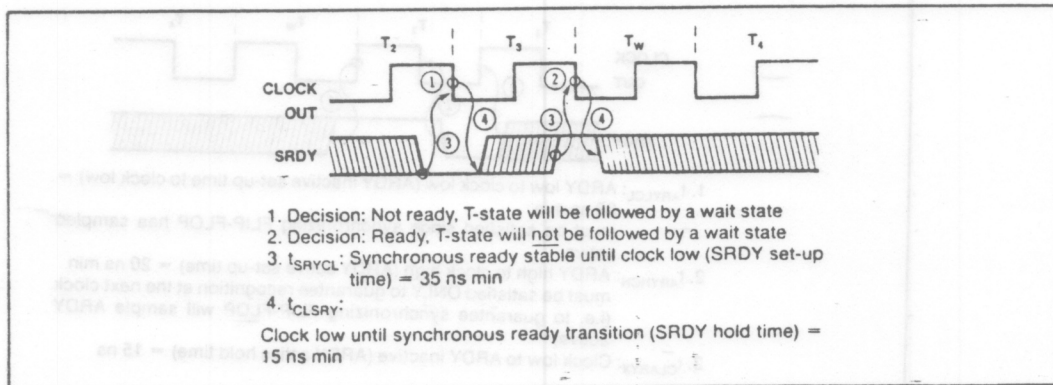


Figure 19. Valid SRDY transitions on the 80186

be immediately followed by T_4 . On the other hand, if the line is sampled inactive at the beginning of either of these two cycles, that cycle will be followed by a T_w . Any asynchronous transition on the $SRDY$ line not occurring at the beginning of T_3 or T_w , that is, when the processor is not "looking at" the ready lines will not cause CPU malfunction.

3.1.10 BUS PERFORMANCE ISSUES

Bus cycles occur sequentially, but do not necessarily come immediately one after another, that is the bus may remain idle for several T states (T_i) between each bus access initiated by the 80186. This occurs whenever the 80186 internal queue is full and no read/write cycles are being requested by the execution unit or integrated DMA unit. The reader should recall that a separate unit, the bus interface unit, fetches opcodes (including immediate data) from memory, while the execution unit actually executes the pre-fetched instructions. The number of clock cycles required to execute an 80186 instruction vary from 2 clock cycles for a register to register move to 67 clock cycles for an integer divide.

If a program contains many long instructions, program execution will be CPU limited, that is, the instruction queue will be constantly filled. Thus, the execution unit does not need to wait for an instruction to be fetched. If a program contains mainly short instructions or data move instructions, the execution will be bus limited. Here, the execution unit will be required to wait often for an instruction to be fetched before it continues its operation. Programs illustrating this effect and performance degradation of each with the addition of wait states are given in appendix G.

All instruction fetches are word (16-bit) fetches from even addresses unless the fetch occurs as a result of a jump to an odd location. This maximizes the utilization

of each bus cycle used for instruction fetching, since each fetch will access two bytes of information. It is also good programming practice to locate all word data at even locations, so that both bytes of the word may be accessed in a single bus cycle (see discussion on data bus interfacing for further information, section 3.1.3 of this note).

Although the amount of bus utilization, i.e., the percentage of bus time used by the 80186 for instruction fetching and execution required for top performance will vary considerably from one program to another, a typical instruction mix on the 80186 will require greater bus utilization than the 8086. This is caused by the higher performance execution unit requiring instructions from the prefetch queue at a greater rate. This also means that the effect of wait states is more pronounced in an 80186 system than in an 8086 system. In all but a few cases, however, the performance degradation incurred by adding a wait state is less than might be expected because instruction fetching and execution are performed by separate units.

3.2 Example Memory Systems

3.2.1 2764 INTERFACE

With the above knowledge of the 80186 bus, various memory interfaces may be generated. One of the simplest of these is the example EPROM interface shown in Figure 20.

The addresses are latched using the address generation circuit shown earlier. Note that the $A0$ line of each EPROM is connected to the $A1$ address line from the 80186, NOT the $A0$ line. Remember, $A0$ only signals a data transfer on the lower 8 bits of the 16-bit data bus! The EPROM outputs are connected directly to the address/data inputs of the 80186, and the 80186 RD signal is used as the OE for the EPROMs.

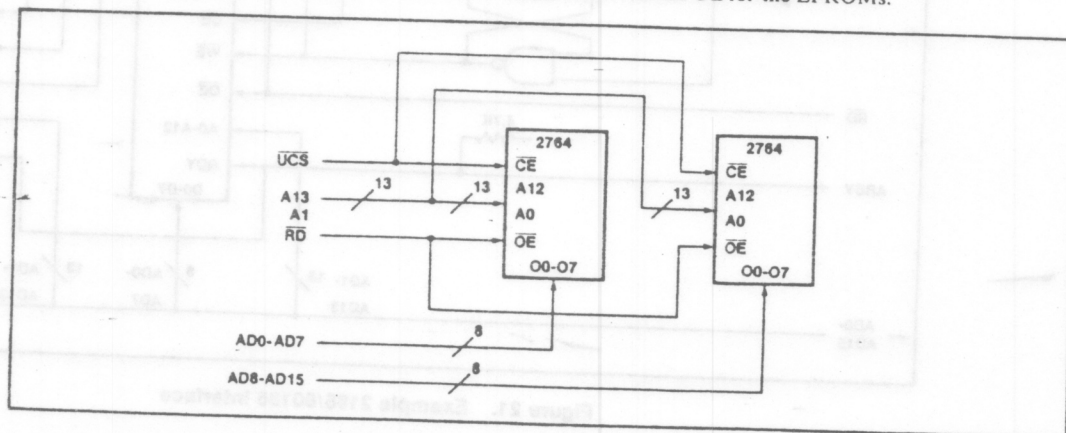


Figure 20. Example 2764/80186 Interface

The chip enable of the EPROM is driven directly by the chip select output of the 80186 (see section 8). In this configuration, the access time calculation for the EPROMs are:

$$\begin{aligned} \text{time from address: } & (3 + N) \cdot t_{\text{CLCL}} - t_{\text{CLAV}} - t_{\text{VOV}}(8282) - t_{\text{DVCL}} \\ & = 375 + (N \cdot 125) - 44 - 30 - 20 \\ & = 281 + (N \cdot 125) \text{ ns} \end{aligned}$$

$$\begin{aligned} \text{time from chip select: } & (3 + N) \cdot t_{\text{CLCL}} - t_{\text{CLCSV}} - t_{\text{DVCL}} \\ & = 375 + (N \cdot 125) - 66 - 20 \\ & = 289 + (N \cdot 125) \text{ ns} \end{aligned}$$

$$\begin{aligned} \text{time from RD (OE): } & (2 + N) \cdot t_{\text{CLCL}} - t_{\text{CLRL}} - t_{\text{DVCL}} \\ & = 250 + (N \cdot 125) - 70 - 20 \\ & = 160 + (N \cdot 125) \text{ ns} \end{aligned}$$

where:

- t_{CLAV} = time from clock low in T_1 until addresses are valid
- t_{CLCL} = clock period of processor
- t_{VOV} = time from input valid of 8282 until output valid of 8282

t_{DVCL} = 186 data valid input setup time until clock low time of T_4

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{CLRL} = time from clock low in T_2 until $\overline{\text{RD}}$ goes low

N = number of wait states inserted

Thus, for 0 wait state operation, 250ns EPROMs must be used. The only significant parameter not included above is t_{RHAV} the time from $\overline{\text{RD}}$ inactive (high) until the 80186 begins driving address information. This parameter is 85ns, which meets the 2764-25 (250ns speed selection) output float time of 85ns. If slower EPROMs are used, a discrete data buffer *MUST* be inserted between the EPROM data lines and the address/data bus, since these devices may continue to drive data information on the multiplexed address/data bus when the 80186 begins to drive address information for the next bus cycle.

3.2.2 2186 INTERFACE

An example interface between the 80186 and 2186 iRAMs is shown in Figure 21. This memory component is almost an ideal match with the 80186, because of its large integration, and its not requiring address latching.

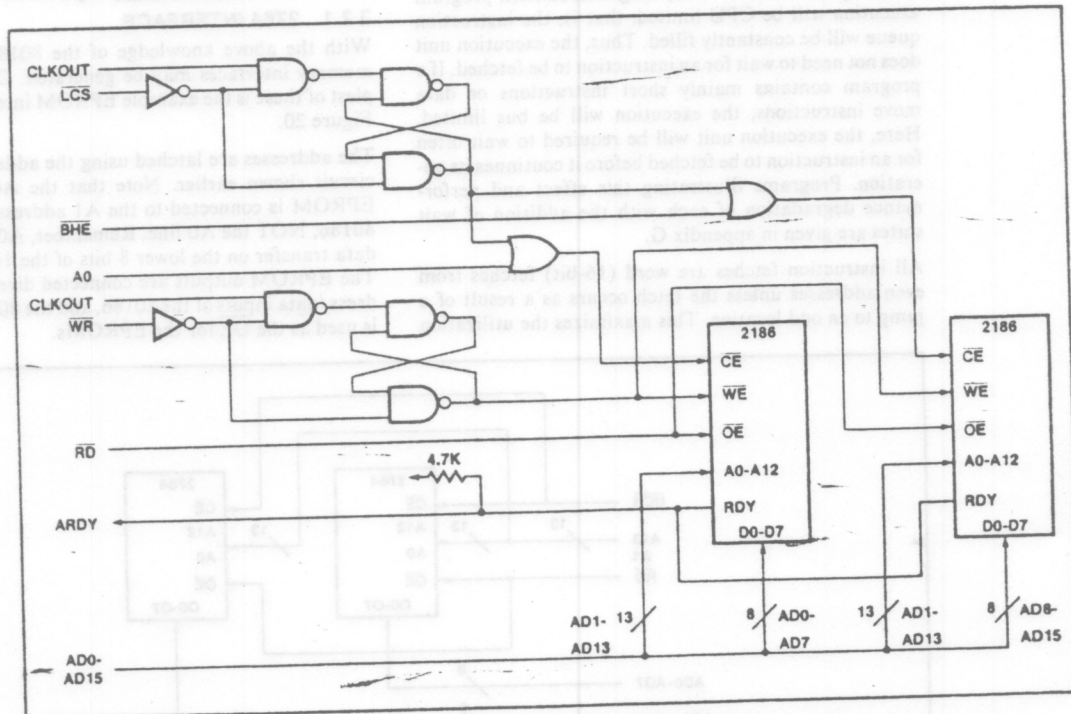


Figure 21. Example 2186/80186 Interface

The 2186 internally is a dynamic RAM integrated with refresh and control circuitry. It operates in two modes, pulse mode and late cycle mode. Pulse mode is entered if the $\overline{CE}$ signal is low to the device a maximum of 130ns, and requires the command input ($\overline{RD}$ or $\overline{WE}$) to go active within 90ns after $\overline{CE}$. Because of these requirements, interfacing the 80186 to the 2186 in pulse mode would be difficult. Instead, the late cycle mode is used. This affords a much simpler interface with no loss of performance. The iRAM automatically selects between these modes by the nature of the control signals.

The 2186 is a leading edge triggered device. This means that address and data information are strobed into the device on the active going (high to low) transition of the command signal. This requires both $\overline{CE}$ and $\overline{WR}$ be delayed until the address and data driven by the 80186 are guaranteed stable. Figure 21 shows a simple circuit which can be used to perform this function. Note that $\overline{ALE}$ CANNOT be used to delay $\overline{CE}$ if addresses are not latched externally, because this would violate the address hold time required by the 2186 (30ns).

Because the 2186s are RAMs, data bus enables ($\overline{BHE}$ and $\overline{A0}$, see previous section) MUST be used to factor either the chip enables or write enables of the lower and upper bytes of the 16-bit RAM memory system. If this is not done, all memory writes, including single byte writes, will write to both the upper and lower bytes of the memory system. The example system shown uses $\overline{BHE}$ and $\overline{A0}$ as factors to the 2186 $\overline{CE}$. This may be done, because both of these signals ($\overline{A0}$ and $\overline{BHE}$) are valid when the address information is valid from the 80186.

The 2186 requires a certain amount of recovery time between its chip enable going inactive and its chip enable going active insure proper operation. For a "normal" cycle (a read or write), this time is $t_{EHEL} = 40ns$. This means that the 80186 chip select lines will go inactive soon enough at the end of a bus cycle to provide the required recovery time even if two consecutive accesses are made to the iRAMs. If the 2186 $\overline{CE}$ is asserted without a command signal ($\overline{WE}$ or $\overline{OE}$), a "False Memory Cycle" (FMC) will be generated. Whenever a FMC is generated, the recovery time is much longer; another memory cycle must not be initiated for 200ns. As a result, if the memory system will generate FMCs, $\overline{CE}$ must be taken away in the middle of the T state (T_3 or T_w) immediately preceding T_4 to insure two consecutive cycles to the iRAM will not violate this parameter. Status going passive (all high) can be used for this purpose. These lines will all go high during the first phase of the next to last T state (either T_3 or T_w) of a bus cycle (see section 3.1.5).

Finally, since it is a dynamic device, the 2186 requires refresh cycles to maintain data integrity. The circuitry to generate these refresh cycles is integrated within the 2186. Because of this, the 2186 has a ready line which is used to suspend processor operation if a processor RAM

access coincides with an internally generated refresh cycle. This is an open collector output, allowing many of them to be wire-OR'ed together, since more than one device may be accessed at a time. These lines are also normally ready, which means that they will be high whenever the 2186 is not being accessed, i.e., they will only be driven low if a processor request coincides with an internal refresh cycle. Thus, the ready lines from the iRAM must be factored into the 80186 RDY circuit only during accesses to the iRAM itself. Since the 2186 refresh logic operates asynchronously to the 80186, this RDY line must be synchronized for proper operation with the 80186, either by the integrated ready synchronizer or by an external circuit. The example circuit uses the integrated synchronizer associated with the ARDY processor input.

The ready lines of the 2186 are active unless a processor access coincides with an internal refresh cycle. These lines must go inactive soon enough after a cycle is requested to insert wait states into the data cycle. The 2186 will drive this line low within 50ns after $\overline{CE}$ is received, which is early enough to force the 80186 to insert wait states if they are required. The primary concern here is that the ARDY line be driven not active before its setup time in the middle of T_2 . This is required by the nature of the asynchronous ready synchronization circuitry of the 80186. Since the RDY pulse of the 2186 may be as narrow as 50ns, if ready was returned after the first stage of the synchronizer, and subsequently changed state within the ready setup and hold time of the high to low going edge of the CPU clock at the end of T_2 , improper operation may occur (see section 3.1.6).

The example interface shown has a zero wait state RAM read access time from $\overline{CE}$ of:

$$\begin{aligned} 3 * t_{CLCL} - t_{CLCSV} - (\text{TTL delay}) - t_{DVCL} \\ = 375 - 66 - 30 - 20 \text{ ns} \\ = 259 \text{ ns} \end{aligned}$$

where:

t_{CLCL} = CPU clock cycle time

t_{CLCSV} = time from clock low in T_1 until chip selects are valid

t_{DVCL} = 80186 data in setup time before clock low in T_2

The data valid delay from $\overline{OE}$ active is less than 100ns, and is therefore not an access time limiter in this interface. Additionally, the 2186 data float time from RD inactive is less than the 85ns 80186 imposed maximum. The $\overline{CE}$ generation circuit shown in Figure 21 provides an address setup time of at least 11ns, and an address hold time of at least 35ns (assuming a maximum two level TTL delay of less than 30ns).

Write cycle address setup and hold times are identical to the read cycle times. The circuit shown provides at least 11ns write data setup and 100ns data hold time from $\overline{WE}$, easily meeting the 0ns setup and 40ns hold times required by the 2186.

For more information concerning 2186 timing and interfacing, please consult the 2186 data sheet, or the application note AP-132, "Designing Memory Systems with the 8Kx8 iRAM" by John Fallin and William Righter (June 1982).

3.2.3 8203 DRAM INTERFACE

An example 8203/DRAM interface is shown in Figure 22. The 8203 provides all required DRAM control signals, address multiplexing, and refresh generation. In this circuit, the 8203 is configured to interface with 64K DRAMs.

All 8203 cycles are generated off control signals ($\overline{RD}$ and $\overline{WR}$) provided by the 80186. These signals will not go active until T_2 of the bus cycle. In addition, since the 8203 clock (generated by the internal crystal oscillator of the 8203) is asynchronous to the 80186 clock, all memory requests by the 80186 must be synchronized to the 8203 before the cycle will be run. To minimize this synchronization time, the 8203 should be used with the highest speed crystal that will maintain DRAM compatibility. Even if a 25 MHz crystal is used (the maximum allowed by the 8203) two wait states will be required by the example circuit when using 150ns DRAMs with an 8 MHz 80186, three wait states if 200ns DRAMs are used (see timing analysis, Figure 23).

The entire RAM array controlled by the 8203 can be selected by one or a group of the 80186 provided chip selects. These chip selects can also be used to insert the wait states required by the interface.

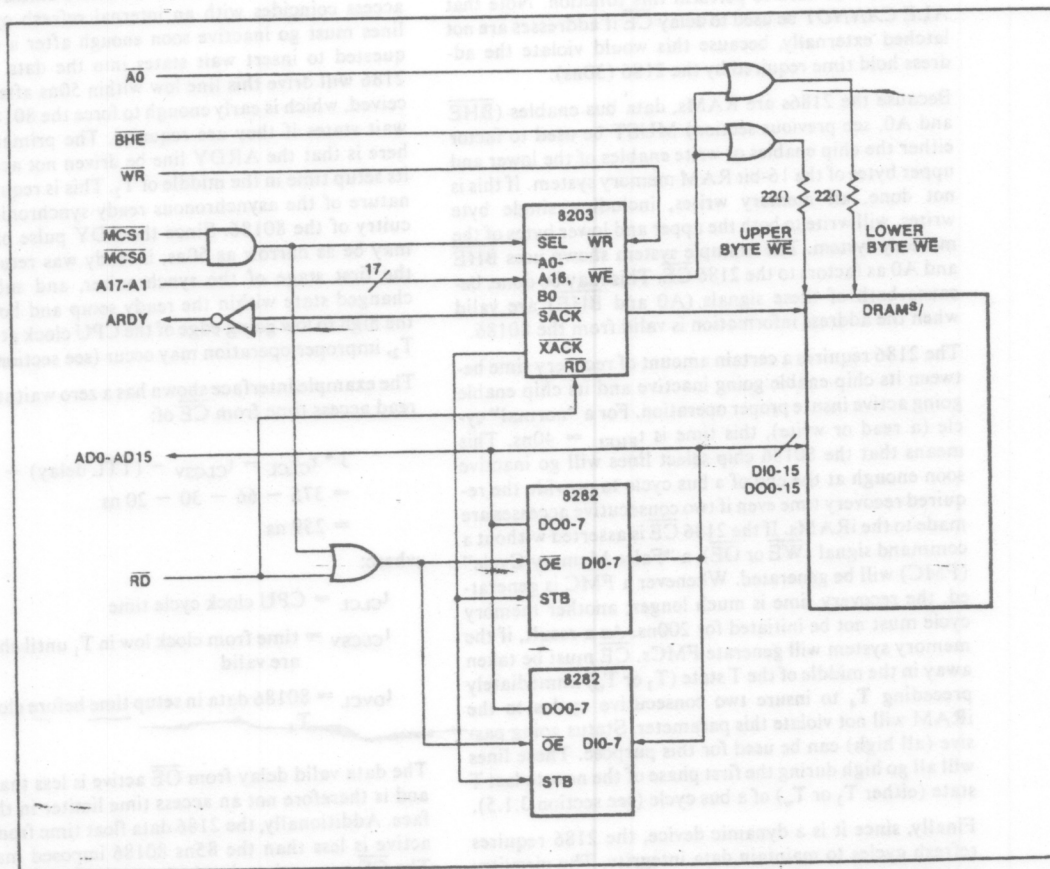


Figure 22. Example 8203/DRAM/80186 Interface

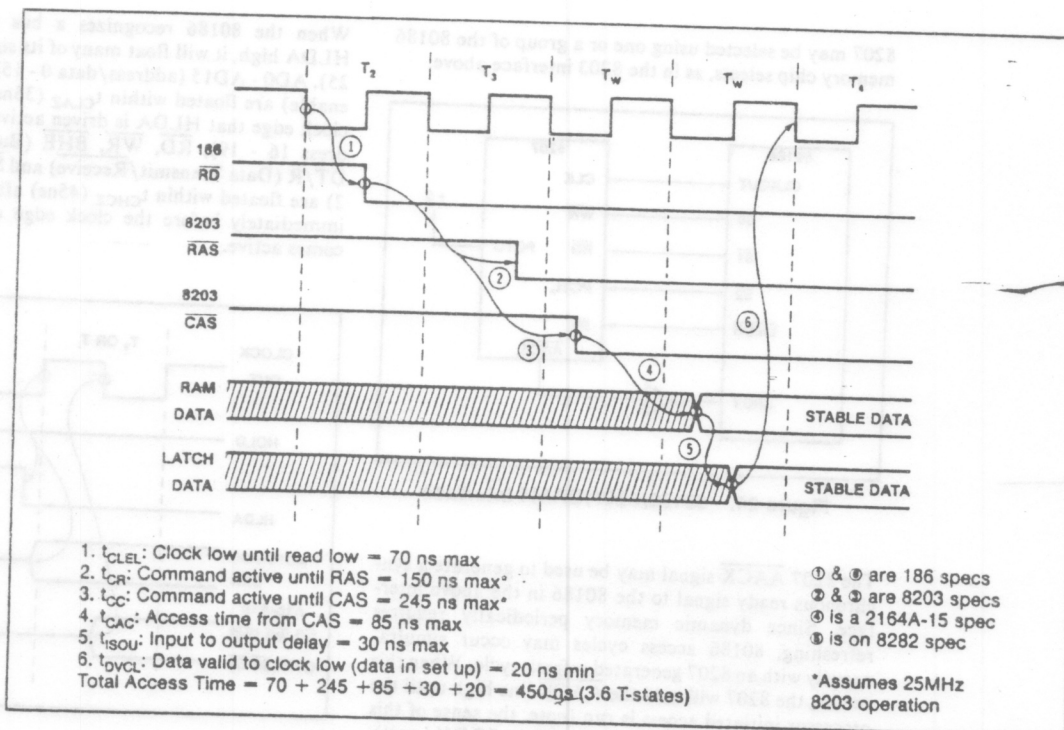


Figure 23. 8203/2164A-15 Access Time Calculation

Since the 8203 is operating asynchronously to the 80186, the RDY output of the 8203 (used to suspend processor operation when a processor DRAM request coincides with a DRAM refresh cycle) must be synchronized to the 80186. The 80186 ARDY line is used to provide the necessary ready synchronization. The 8203 ready outputs operate in a normally not ready mode, that is, they are only driven active when an 8203 cycle is being executed, and a refresh cycle is not being run. This is fundamentally different than the normally ready mode used by the 2186 iRAMs (see previous section). The 8203 SACK signal is presented to the 80186 only when the DRAM is being accessed. Notice that the SACK output of the 8203 is used, rather than the XACK output. Since the 80186 will insert at least one full CPU clock cycle between the time RDY is sampled active, and the time data must be present on the data bus, using the XACK signal would insert unnecessary additional wait states, since it does not indicate ready until valid data is available from the memory.

For more information about 8203/DRAM interfacing and timing, please consult the 8203 data sheet, or AP97A, "Interfacing Dynamic RAM to iAPX86/88

Systems Using the Intel 8202A and 8203" by Brad May (April 1982).

3.2.4 8207 DRAM INTERFACE

The 8207 advanced dual-port DRAM controller provides a high performance DRAM memory interface specifically for 80186 or 80286 microcomputer systems. This controller provides all address-multiplexing and DRAM refresh circuitry. In addition, it synchronizes and arbitrates memory requests from two different ports (e.g., an 80186 and a Multibus), allowing the two ports to share memory. Finally, the 8207 provides a simple interface to the 8206 error detection and correction chip.

The simplest 8207 (and also the highest performance) interface is shown in Figure 24. This shows the 80186 connected to an 8207 using the 8207 slow cycle, synchronous status interface. In this mode, the 8207 decodes the type of cycle to be run directly from the status lines of the 80186. In addition, since the 8207 CLOCKIN is driven by the CLOCKOUT of the 80186, any performance degradation caused by required memory request synchronization between the 80186 and the 8207 is not present. Finally, the entire memory array driven by the

8207 may be selected using one or a group of the 80186 memory chip selects, as in the 8203 interface above.

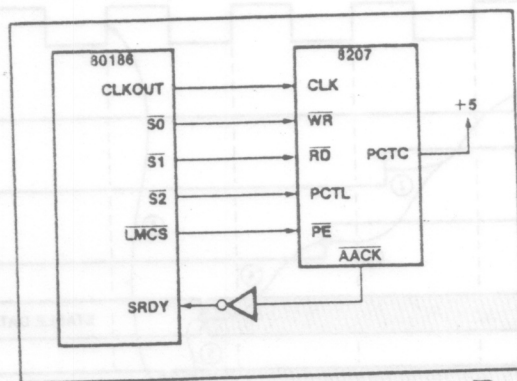


Figure 24. 80186/8207/DRAM interface

The 8207 AACK signal may be used to generate a synchronous ready signal to the 80186 in the above interface. Since dynamic memory periodically requires refreshing, 80186 access cycles may occur simultaneously with an 8207 generated refresh cycle. When this occurs, the 8207 will hold the AACK line high until the processor initiated access is run (note, the sense of this line is reversed with respect to the 80186 SRDY input). This signal should be factored with the DRAM (8207) select input and used to drive the SRDY line of the 80186. Remember that only one of SRDY and ARDY needs to be active for a bus cycle to be terminated. If asynchronous devices (e.g., a Multibus interface) are connected to the ARDY line with the 8207 connected to the SRDY line, care must be taken in design of the ready circuit such that only one of the RDY lines is driven active at a time to prevent premature termination of the bus cycle.

3.3 HOLD/HLDA Interface

The 80186 employs a HOLD/HLDA bus exchange protocol. This protocol allows other asynchronous bus master devices (i.e., ones which drive address, data, and control information on the bus) to gain control of the bus to perform bus cycles (memory or I/O reads or writes).

3.3.1 HOLD RESPONSE

In the HOLD/HLDA protocol, a device requiring bus control (e.g., an external DMA device) raises the HOLD line. In response to this HOLD request, the 80186 will raise its HLDA line after it has finished its current bus activity. When the external device is finished with the bus, it drops its bus HOLD request. The 80186 responds by dropping its HLDA line and resuming bus operation.

When the 80186 recognizes a bus hold by driving HLDA high, it will float many of its signals (see Figure 25). AD0 - AD15 (address/data 0 - 15) and DEN (data enable) are floated within t_{CLAZ} (35ns) after the same clock edge that HLDA is driven active. A16-A19 (address 16 - 19), RD, WR, BHE (Bus High Enable), DT/R (Data Transmit/Receive) and S0 - S2 (status 0 - 2) are floated within t_{CHCZ} (45ns) after the clock edge immediately before the clock edge on which HLDA comes active.

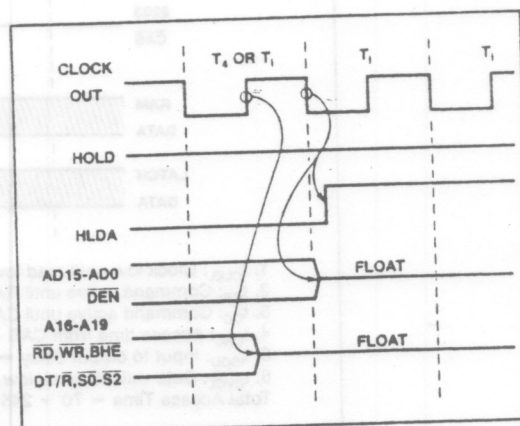


Figure 25. Signal Float/HLDA Timing of the 80186

Only the above mentioned signals are floated during bus HOLD. Of the signals not floated by the 80186, some have to do with peripheral functionality (e.g., TmrOut). Many others either directly or indirectly control bus devices. These signals are ALE (Address Latch Enable, see section 3.1.2) and all the chip select lines (UCS, LCS, MCS0-3, and PCS0-6). The designer must be aware that the chip select circuitry does not look at externally generated addresses (see section 10 for a discussion of the chip select logic). Thus, for memory or peripheral devices which are addressed by external bus master devices, discrete chip select and ready generation logic must be used.

3.3.2 HOLD/HLDA TIMING AND BUS LATENCY

The time required between HOLD going active and the 80186 driving HLDA active is known as bus latency. Many factors affect this latency, including synchronization delays, bus cycle times, locked transfer times and interrupt acknowledge cycles.

The HOLD request line is internally synchronized by the 80186, and may therefore be an asynchronous signal. To guarantee recognition on a certain clock edge, it must satisfy a certain setup and hold time to the falling

edge of the CPU clock. A full CPU clock cycle is required for this synchronization, that is, the internal HOLD signal is not presented to the internal bus arbitration circuitry until one full clock cycle after it is latched from the HOLD input (see Appendix B for a dis-

cussion of 80186 synchronizers). If the bus is idle, HLDA will follow HOLD by two CPU clock cycles plus a small amount of setup and propagation delay time. The first clock cycle synchronizes the input; the second signals the internal circuitry to initiate a bus hold. (see Figure 26).

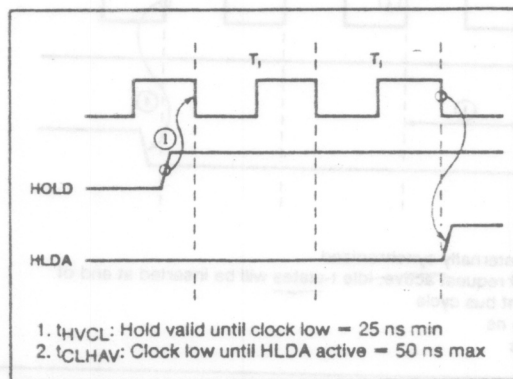


Figure 26. 80186 Idle Bus Hold/HLDA Timing

Many factors influence the number of clock cycles between a HOLD request and a HLDA. These may make bus latency longer than the best case shown above. Perhaps the most important factor is that the 80186 will not relinquish the local bus until the bus is idle. An idle bus occurs whenever the 80186 is not performing any bus transfers. As stated in section 3.1.1, when the bus is idle, the 80186 generates idle T-states. The bus can become idle only at the end of a bus cycle. Thus, the 80186 can recognize HOLD only after the end of its current bus cycle. The 80186 will normally insert no T_1 states between T_4 and T_1 of the next bus cycle if it requires any bus activity (e.g., instruction fetches or I/O reads). However, the 80186 may not have an immediate need for the bus after a bus cycle, and will insert T_1 states independent of the HOLD input (see section 3.1.7).

When the HOLD request is active, the 80186 will be

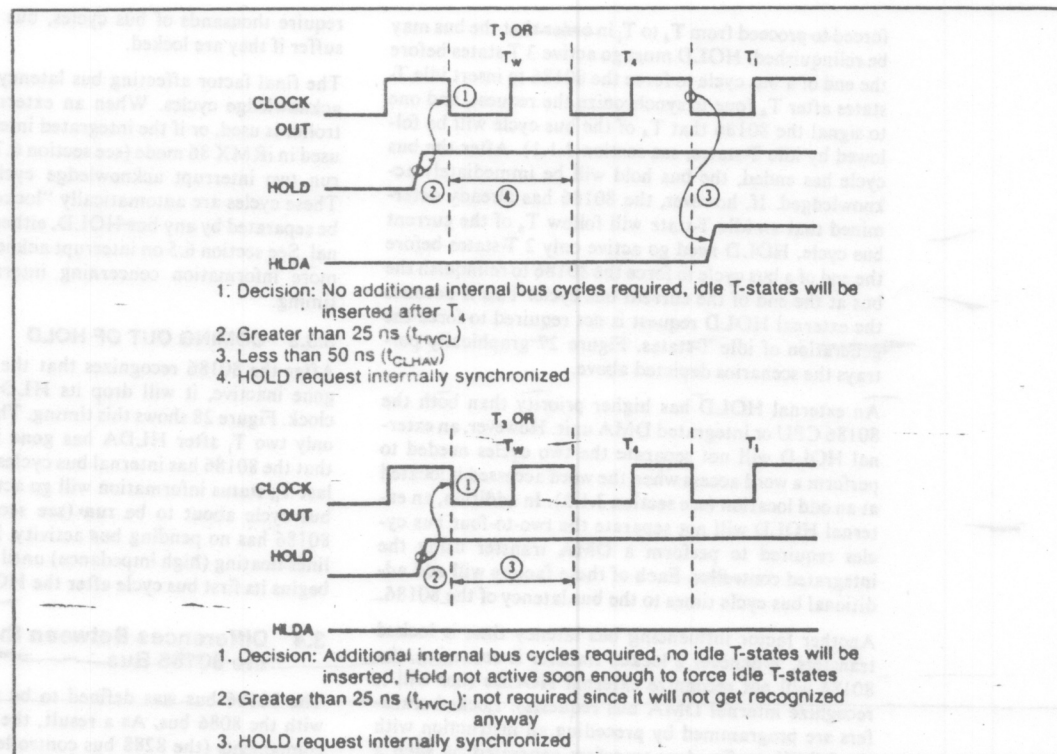


Figure 27. HOLD/HLDA Timing in the 80186

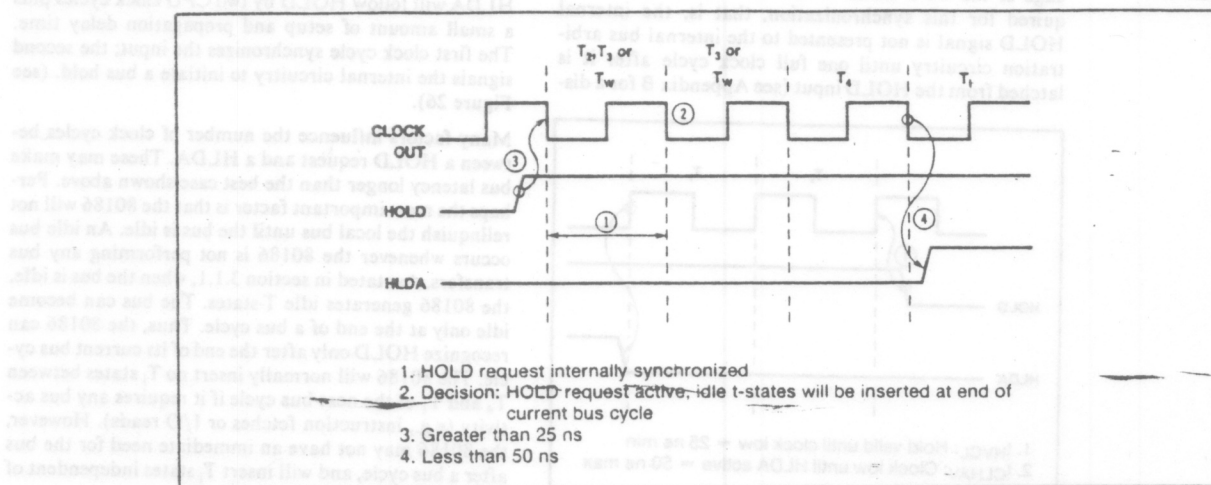


Figure 27A. HOLD/HLDA Timing in the 80186

forced to proceed from T_4 to T_1 in order that the bus may be relinquished. HOLD must go active 3 T-states before the end of a bus cycle to force the 80186 to insert idle T-states after T_4 (one to synchronize the request, and one to signal the 80186 that T_4 of the bus cycle will be followed by idle T-states, see section 3.1.1). After the bus cycle has ended, the bus hold will be immediately acknowledged. If, however, the 80186 has already determined that an idle T-state will follow T_4 of the current bus cycle, HOLD need go active only 2 T-states before the end of a bus cycle to force the 80186 to relinquish the bus at the end of the current bus cycle. This is because the external HOLD request is not required to force the generation of idle T-states. Figure 27 graphically portrays the scenarios depicted above.

An external HOLD has higher priority than both the 80186 CPU or integrated DMA unit. However, an external HOLD will not separate the two cycles needed to perform a word access when the word accessed is located at an odd location (see section 3.1.3). In addition, an external HOLD will not separate the two-to-four bus cycles required to perform a DMA transfer using the integrated controller. Each of these factors will add additional bus cycle times to the bus latency of the 80186.

Another factor influencing bus latency time is locked transfers. Whenever a locked transfer is occurring, the 80186 will not recognize external HOLDs (nor will it recognize internal DMA bus requests). Locked transfers are programmed by preceding an instruction with the LOCK prefix. Any transfers generated by such a prefixed instruction will be locked, and will not be separated by any external bus requesting device. String instructions may be locked. Since string transfers may

require thousands of bus cycles, bus latency time will suffer if they are locked.

The final factor affecting bus latency time is interrupt acknowledge cycles. When an external interrupt controller is used, or if the integrated interrupt controller is used in iRMX 86 mode (see section 6.7.4) the 80186 will run two interrupt acknowledge cycles back to back. These cycles are automatically "locked" and will never be separated by any bus HOLD, either internal or external. See section 6.5 on interrupt acknowledge timing for more information concerning interrupt acknowledge timing.

3.3.3 COMING OUT OF HOLD

After the 80186 recognizes that the HOLD input has gone inactive, it will drop its HLDA line in a single clock. Figure 28 shows this timing. The 80186 will insert only two T_1 after HLDA has gone inactive, assuming that the 80186 has internal bus cycles to run. During the last T_1 , status information will go active concerning the bus cycle about to be run (see section 3.1.1). If the 80186 has no pending bus activity, it will maintain all lines floating (high impedance) until the last T_1 before it begins its first bus cycle after the HOLD.

3.4 Differences Between the 8086 bus and the 80186 Bus

The 80186 bus was defined to be upward compatible with the 8086 bus. As a result, the 8086 bus interface components (the 8288 bus controller and the 8289 bus arbiter) may be used directly with the 80186. There are a few significant differences between the two processors which should be considered.

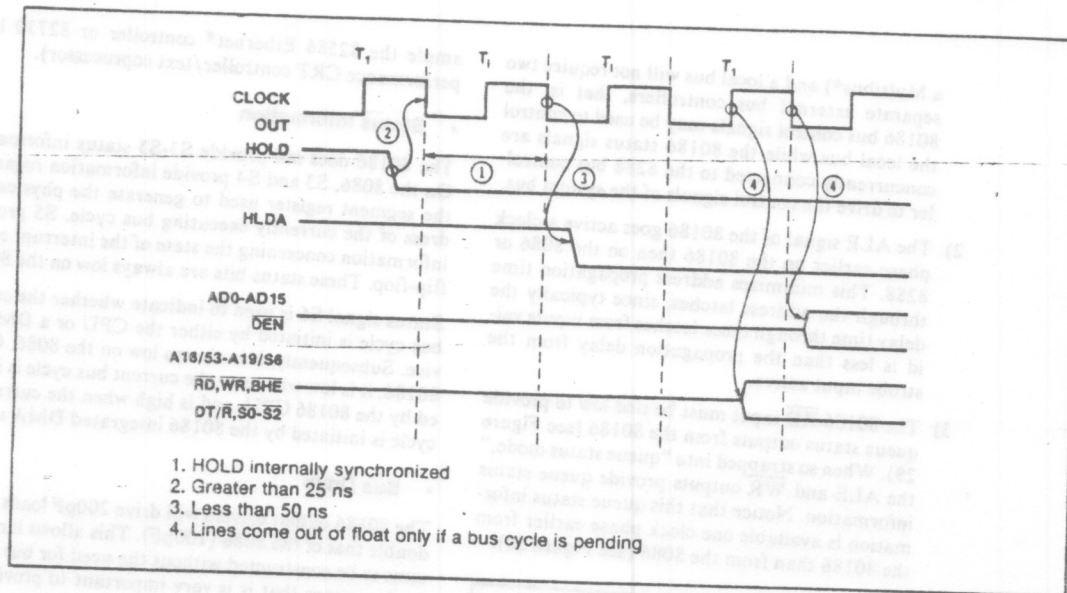


Figure 28. 80186 Coming out of Hold

• CPU Duty Cycle and Clock Generator

The 80186 employs an integrated clock generator which provides a 50% duty cycle CPU clock (1/2 of the time it is high, the other 1/2 of the time it is low). This is different than the 8086, which employs an external clock generator (the 8284A) with a 33% duty cycle CPU clock (1/3 of the time it is high, the other 2/3 of the time, it is low). These differences manifest themselves as follows:

- 1) No oscillator output is available from the 80186, as it is available from the 8284A clock generator.
- 2) The 80186 does not provide a PCLK (50% duty cycle, 1/2 CPU clock frequency) output as does the 8284A.
- 3) The clock low phase of the 80186 is narrower, and the clock high phase is wider than on the same-speed 8086.
- 4) The 80186 does not internally factor AEN with RDY. This means that if both RDY inputs (ARDY and SRDY) are used, external logic must be used to prevent the RDY not connected to a certain device from being driven active during an access to this device (remember, only one RDY input needs to be active to terminate a bus cycle, see section 3.1.6).
- 5) The 80186 concurrently provides both a single asynchronous ready input and a single synchronous ready input, while the 8284A provides ei-

ther two synchronous ready inputs or two asynchronous ready inputs as a user strapable option.

- 6) The CLOCKOUT (CPU clock output signal) drive capacity of the 80186 is less than the CPU clock drive capacity of the 8284A. This means that not as many high speed devices (e.g., Schottky TTL flip-flops) may be connected to this signal as can be used with the 8284A clock output.
- 7) The crystal or external oscillator used by the 80186 is twice the CPU clock frequency, while the crystal or external oscillator used with the 8284A is three times the CPU clock frequency.

• Local Bus Controller and Control Signals

The 80186 simultaneously provides both local bus controller outputs ($\overline{RD}$, $\overline{WR}$, $\overline{ALE}$, $\overline{DEN}$ and $\overline{DT/R}$) and status outputs (S_0 , S_1 , S_2) for use with the 8288 bus controller. This is different from the 8086 where the local bus controller outputs (generated only in min mode) are sacrificed if status outputs (generated only in max mode) are desired. These differences will manifest themselves in 8086 systems and 80186 systems as follows:

- 1) Because the 80186 can simultaneously provide local bus control signals and status outputs, many systems supporting both a system bus (e.g.,

a Multibus®) and a local bus will not require two separate external bus controllers, that is, the 80186 bus control signals may be used to control the local bus while the 80186 status signals are concurrently connected to the 8288 bus controller to drive the control signals of the system bus.

- 2) The ALE signal of the 80186 goes active a clock phase earlier on the 80186 than on the 8086 or 8288. This minimizes address propagation time through the address latches, since typically the delay time through these latches from inputs valid is less than the propagation delay from the strobe input active.
- 3) The 80186 $\overline{RD}$ input must be tied low to provide queue status outputs from the 80186 (see Figure 29). When so strapped into "queue status mode," the ALE and $\overline{WR}$ outputs provide queue status information. Notice that this queue status information is available one clock phase earlier from the 80186 than from the 8086 (see Figure 30).

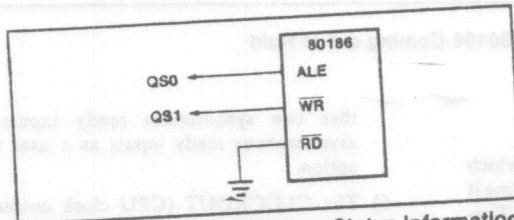


Figure 29. Generating Queue Status Information from the 80186

• HOLD/HLDA vs. RQ/GT

As discussed earlier, the 80186 uses a HOLD/HLDA type of protocol for exchanging bus mastership (like the 8086 in min mode) rather than the RQ/GT protocol used by the 8086 in max mode. This allows compatibility with Intel's the new generation of high performance/high integration bus master peripheral devices (for ex-

ample the 82586 Ethernet® controller or 82730 high performance CRT controller/text coprocessor).

• Status Information

The 80186 does not provide S3-S5 status information. On the 8086, S3 and S4 provide information regarding the segment register used to generate the physical address of the currently executing bus cycle. S5 provides information concerning the state of the interrupt enable flip-flop. These status bits are always low on the 80186.

Status signal S6 is used to indicate whether the current bus cycle is initiated by either the CPU or a DMA device. Subsequently, it is always low on the 8086. On the 80186, it is low whenever the current bus cycle is initiated by the 80186 CPU, and is high when the current bus cycle is initiated by the 80186 integrated DMA unit.

• Bus Drive

The 80186 output drivers will drive 200pF loads. This is double that of the 8086 (100pF). This allows larger systems to be constructed without the need for bus buffers. It also means that it is very important to provide good grounds to the 80186, since its large drivers can discharge its outputs very quickly causing large current transients on the 80186 ground pins.

• Misc.

The 80186 does not provide early and late write signals, as does the 8288 bus controller. The $\overline{WR}$ signal generated by the 80186 corresponds to the early write signal of the 8288. This means that data is not stable on the address/data bus when this signal is driven active.

The 80186 also does not provide differentiated I/O and memory read and write command signals. If these signals are desired, an external 8288 bus controller may be used, or the $\overline{S2}$ signal may be used to synthesize differentiated commands (see section 3.1.4).

*Ethernet is a registered trademark of Xerox Corp.

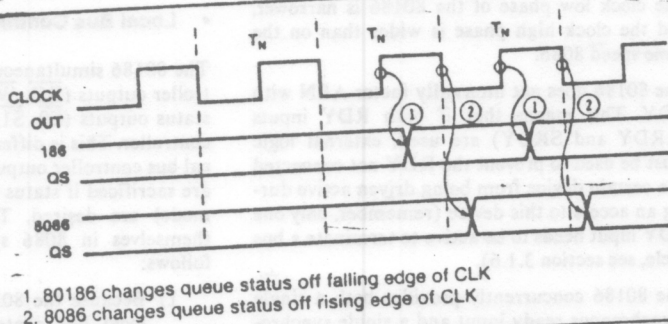


Figure 30. 80186 and 8086 Queue Status Generation

4. DMA UNIT INTERFACING

The 80186 includes a DMA unit which provides two independent high speed DMA channels. These channels operate independently of the CPU, and drive all integrated bus interface components (bus controller, chip selects, etc.) exactly as the CPU (see Figure 31). This means that bus cycles initiated by the DMA unit are exactly the same as bus cycles initiated by the CPU (except that $S6 = 1$ during all DMA initiated cycles, see section 3.1). Thus interfacing with the DMA unit itself is very simple, since except for the addition of the DMA request connection, it is exactly the same as interfacing to the CPU.

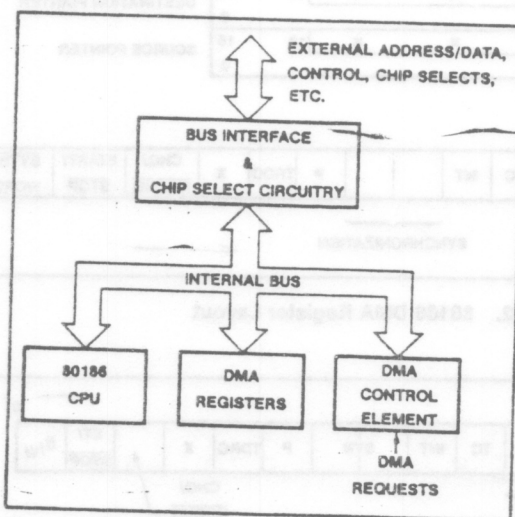


Figure 31. 80186 CPU/DMA Channel Internal Model

4.1 DMA Features

Each of the two DMA channels provides the following features:

- Independent 20-bit source and destination pointers which are used to access the I/O or memory location from which data will be fetched or to which data will be deposited
- Programmable auto-increment, auto-decrement or neither of the source and destination pointers after each DMA transfer
- Programmable termination of DMA activity after a certain number of DMA transfers
- Programmable CPU interruption at DMA termination
- Byte or word DMA transfers to or from even or odd memory or I/O addresses

• Programmable generation of DMA requests by:

- 1) the source of the data
- 2) the destination of the data
- 3) timer 2 (see section 5)
- 4) the DMA unit itself (continuous DMA requests)

4.2 DMA Unit Programming

Each of the two DMA channels contains a number of registers which are used to control channel operation. These registers are included in the 80186 integrated peripheral control block (see appendix A). These registers include the source and destination pointer registers, the transfer count register and the control register. The layout and interpretation of the bits in these registers is given in Figure 32.

The 20-bit source and destination pointers allow access to the complete 1 Mbyte address space of the 80186, and that all 20 bits are affected by the auto-increment or auto-decrement unit of the DMA (i.e., the DMA channels address the full 1 Mbyte address space of the 80186 as a flat, linear array without segments). When addressing I/O space, the upper 4 bits of the DMA pointer registers should be programmed to be 0. If they are not programmed 0, then the programmed value (greater than 64K in I/O space) will be driven onto the address bus (an area of I/O space not accessible to the CPU). The data transfer will occur correctly, however.

After every DMA transfer the 16-bit DMA transfer count register it is decremented by 1, whether a byte transfer or a word transfer has occurred. If the TC bit in the DMA control register is set, the DMA ST/STOP bit (see below) will be cleared when this register goes to 0, causing all DMA activity to cease. A transfer count of zero allows 65536 (2^{16}) transfers.

The DMA control register (see Figure 33) contains bits which control various channel characteristics, including for each of the data source and destination whether the pointer points to memory or I/O space, or whether the pointer will be incremented, decremented or left alone after each DMA transfer. It also contains a bit which selects between byte or word transfers. Two synchronization bits are used to determine the source of the DMA requests (see section 4.7). The TC bit determines whether DMA activity will cease after a programmed number of DMA transfers, and the INT bit is used to enable interrupts to the processor when this has occurred (note that an interrupt will not be generated to the CPU when the transfer count register reaches zero unless both the INT bit and the TC bit are set).

The control register also contains a start/stop (ST/STOP) bit. This bit is used to enable DMA transfers. Whenever this bit is set, the channel is



All DMA unit programming registers are directly accessible by the CPU. This means the CPU can, for example, modify the DMA source pointer register after 137 DMA transfers have occurred, and have the new pointer value used for the 138th DMA transfer. If more than one register in the DMA channel is being modified at any time that a DMA request may be generated and the DMA channel is enabled (the ST/STOP bit in the control register is set), the register programming values should be placed in memory locations and moved into the DMA registers using a locked string move instruction. This will prevent a DMA transfer from occurring after only half of the register values have changed. The above also holds true if a read/modify/write type of operation is being performed (e.g., ANDing off bits in a pointer register in a single AND instruction to a pointer register mapped into memory space).

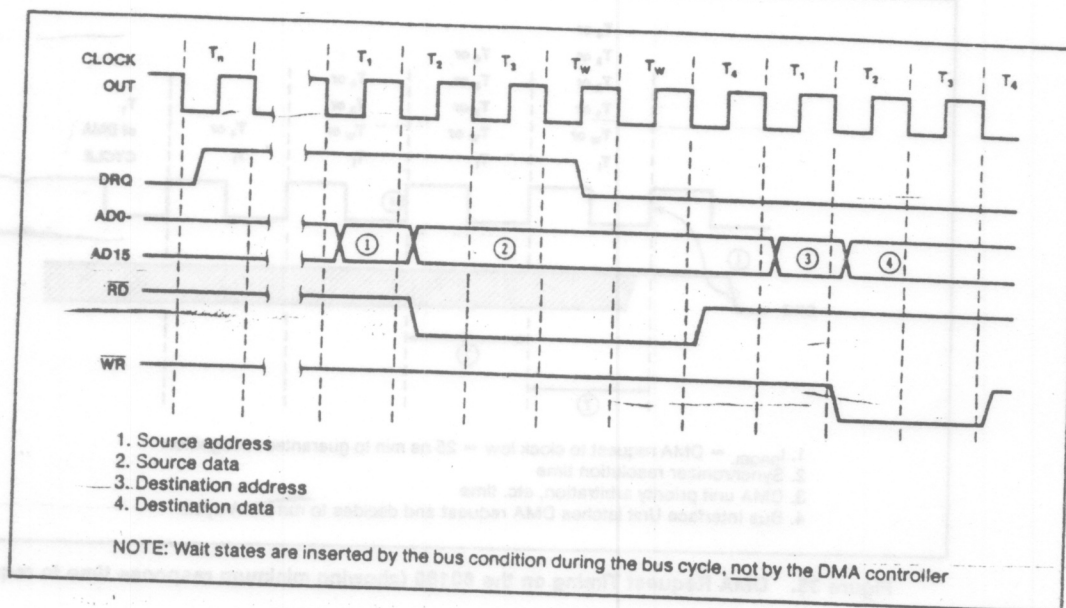


Figure 34. Example DMA Transfer Cycle on the 80186

4.3 DMA Transfers

Every DMA transfer in the 80186 consists of two independent bus cycles, the fetch cycle and the deposit cycle (see Figure 34). During the fetch cycle, the byte or word data is accessed from memory or I/O space using the address in the source pointer register. The data accessed is placed in an internal temporary register, which is not accessible to the CPU. During the deposit cycle, the byte or word data in this internal register is placed in memory or I/O space using the address in the destination pointer register. These two bus cycles will not be separated by bus HOLD or by the other DMA channel, and one will never be run without the other except when the CPU is RESET. Notice that the bus cycles run by the DMA unit are exactly the same as memory or I/O bus cycles run by the CPU. The only difference between the two is the state of the S6 status line (which is multiplexed on the A19 line): on all CPU initiated bus cycles, this status line will be driven low; on all DMA initiated bus cycles, this status line will be driven high.

4.4 DMA Requests

Each DMA channel has a single DMA request line by which an external device may request a DMA transfer. The synchronization bits in the DMA control register determine whether this line is interpreted to be connected to the source of the DMA data or the destination of the DMA data. All transfer requests on this line are synchronized to the CPU clock before being presented to internal

DMA logic. This means that any asynchronous transitions of the DMA request line will not cause the DMA channel to malfunction. In addition to external requests, DMA requests may be generated whenever the internal timer 2 times out, or continuously by programming the synchronization bits in the DMA control register to call for unsynchronized DMA transfers.

4.4.1 DMA REQUEST TIMING AND LATENCY

Before any DMA request can be generated, the 80186 internal bus must be granted to the DMA unit. A certain amount of time is required for the CPU to grant this internal bus to the DMA unit. The time between a DMA request being issued and the DMA transfer being run is known as DMA latency. Many of the issues concerning DMA latency are the same as those concerning bus latency (see section 3.3.2). The only important difference is that external HOLD always has bus priority over an internal DMA transfer. Thus, the latency time of an internal DMA cycle will suffer during an external bus HOLD.

Each DMA channel has a programmed priority relative to the other DMA channel. Both channels may be programmed to be the same priority, or one may be programmed to be of higher priority than the other channel. If both channels are active, DMA latency will suffer on the lower priority channel. If both channels are active and both channels are of the same programmed priority, DMA-transfer cycles will alternate between the two channels (i.e., the first channel will perform a fetch and



The minimum timing required to generate a DMA cycle is shown in Figure 35. Note that the minimum time from DRQ becoming active until the beginning of the first DMA cycle is 4 CPU clock cycles, that is, a DMA request is sampled 4 clock cycles before the beginning of a bus cycle to determine if any DMA activity will be required. This time is independent of the number of wait states inserted in the bus cycle. The maximum DMA latency is a function of other processor activity (see above).

Also notice that if DRQ is sampled active at 1 in Figure 35, the DMA cycle will be executed, even if the DMA request goes inactive before the beginning of the first DMA cycle. This does not mean that the DMA request is latched into the processor such that any transition on the DMA request line will cause a DMA cycle eventually. Quite the contrary, DMA request must be active at a certain time before the end of a bus cycle for the DMA request to be recognized by the processor. If the DMA request line goes inactive before that window, then no DMA cycles will be run.



4.5 DMA Acknowledge

The 80186 generates no explicit DMA acknowledge signal. Instead, the 80186 performs a read or write directly to the DMA requesting device. If required, a DMA acknowledge signal can be generated by a decode of an address, or by merely using one of the PCS lines (see Figure 36). Note ALE must be used to factor the DACK because addresses are not guaranteed stable when chip selects go active. This is required because if the address is not stable when the PCS goes active, glitches can occur at the output of the DACK generation circuitry as the address lines change state. Once ALE has gone low, the addresses are guaranteed to have been stable for at least t_{AVAL} (30ns).

4.6 Internally Generated DMA Requests

There are two types in internally synchronized DMA transfers, that is, transfer initiated by a unit integrated in the 80186. These two types are transfers in which the DMA request is generated by timer 2, or where DMA request is generated by the DMA channel itself.

The DMA channel can be programmed such that whenever timer 2 reaches its maximum count, a DMA request will be generated. This feature is selected by setting the TDRQ bit in the DMA channel control register. A DMA request generated in this manner will be latched in the DMA controller, so that once the timer request has been generated, it cannot be cleared except by running the DMA cycle or by clearing the TDRQ bits in both DMA control registers. Before any DMA requests are generated in this mode, timer 2 must be initiated and enabled.

A timer requested DMA cycle being run by either DMA channel will reset the timer request. Thus, if both channels are using it to request a DMA cycle, only one DMA channel will execute a transfer for every timeout of timer 2. Another implication of having a single bit timer DMA request latch in the DMA controller is that if another timer 2 timeout occurs before a DMA channel has a chance to run a DMA transfer, the first request will be lost, i.e., only a single DMA transfer will occur, even though the timer has timed out twice.

The DMA channel can also be programmed to provide its own DMA requests. In this mode, DMA transfer cycles will be run continuously at the maximum bus bandwidth, one after the other until the preprogrammed number of DMA transfers (in the DMA transfer count register) have occurred. This mode is selected by programming the synchronization bits in the DMA control register for unsynchronized transfers. Note that in this mode, the DMA controller will monopolize the CPU bus, i.e., the CPU will not be able to perform opcode fetching, memory operations, etc., while the DMA transfers are occurring. Also notice that the DMA will only perform the number of transfers indicated in the

maximum count register regardless of the state of the TC bit in the DMA control register.

4.7 Externally Synchronized DMA Transfers

There are two types of externally synchronized DMA transfers, that is, DMA transfers which are requested by an external device rather than by integrated timer 2 or by the DMA channel itself (in unsynchronized transfers). These are source synchronized and destination synchronized transfers. These modes are selected by programming the synchronization bits in the DMA channel control register. The only difference between the two is the time at which the DMA request pin is sampled to determine if another DMA transfer is immediately required after the currently executing DMA transfer. On source synchronized transfers, this is done such that two source synchronized DMA transfers may occur one immediately after the other, while on destination synchronized transfers a certain amount of idle time is automatically inserted between two DMA transfers to allow time for the DMA requesting device to drive its DMA request inactive.

4.7.1 SOURCE SYNCHRONIZED DMA TRANSFERS

In a source synchronized DMA transfer, the source of the DMA data requests the DMA cycle. An example of this would be a floppy disk read from the disk to main memory. In this type of transfer, the device requesting the transfer is read during the fetch cycle of the DMA transfer. Since it takes 4 CPU clock cycles from the time DMA request is sampled to the time the DMA transfer is actually begun, and a bus cycle takes a minimum of 4 clock cycles, the earliest time the DMA request pin will be sampled for another DMA transfer will be at the beginning of the deposit cycle of a DMA transfer. This allows over 3 CPU clock cycles between the time the DMA requesting device receives an acknowledge to its DMA request (around the beginning of T_2 of the DMA fetch cycle), and the time it must drive this request inactive (assuming no wait states) to insure that another DMA transfer is not performed if it is not desired (see Figure 37).

4.7.2 DESTINATION SYNCHRONIZED DMA TRANSFERS

In destination synchronized DMA transfers, the destination of the DMA data requests the DMA transfer. An example of this would be a floppy disk write from main memory to the disk. In this type of transfer, the device requesting the transfer is written during the deposit cycle of the DMA transfer. This causes a problem, since the DMA requesting device will not receive notification of the DMA cycle being run until 3 clock cycles before the end of the DMA transfer (if no wait states are being

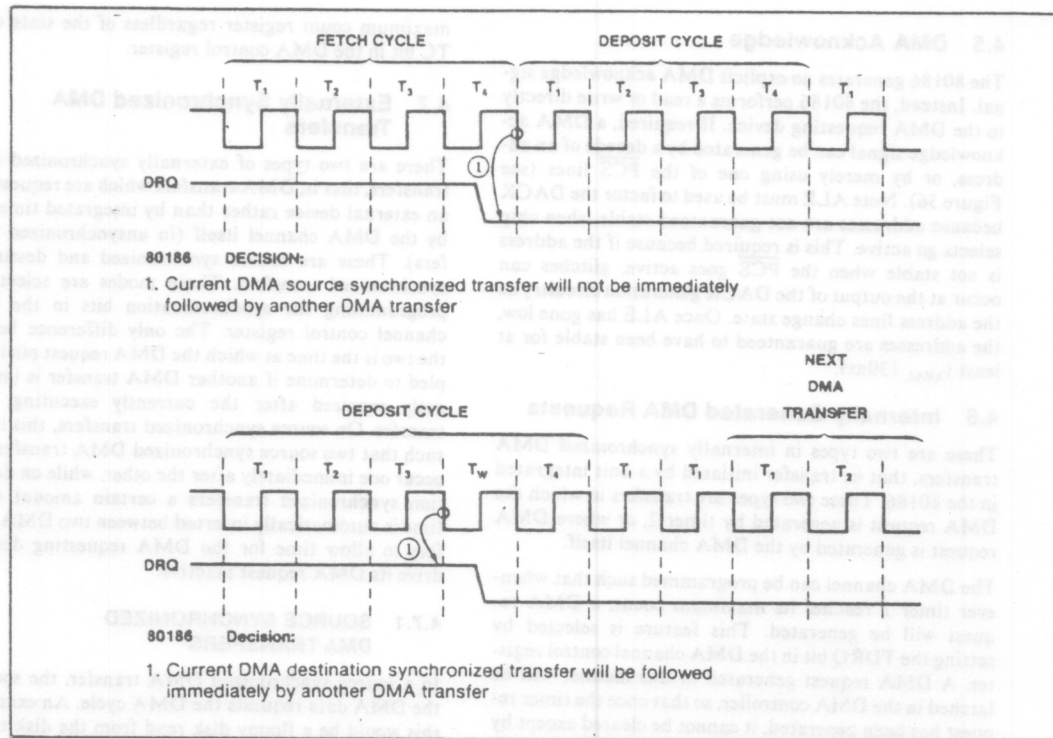


Figure 37. Source & Destination Synchronized DMA Request Timing

inserted into the deposit cycle of the DMA transfer) and it takes 4 clock cycles to determine whether another DMA cycle should be run immediately following the current DMA transfer. To get around this problem, the DMA unit will relinquish the CPU bus after each destination synchronized DMA transfer for at least 2 CPU clock cycles to allow the DMA requesting device time to drop its DMA request if it does not immediately desire another immediate DMA transfer. When the bus is relinquished by the DMA unit, the CPU may resume bus operation (e.g., instruction fetching, memory or I/O reads or writes, etc.) Thus, typically, a CPU initiated bus cycle will be inserted between each destination synchronized DMA transfer. If no CPU bus activity is required, however (and none can be guaranteed), the DMA unit will insert only 2 CPU clock cycles between the deposit cycle of one DMA transfer and the fetch cycle of the next DMA transfer. This means that the DMA destination requesting device must drop its DMA request at least two clock cycles before the end of the deposit cycle regardless of the number of wait states inserted into the bus cycle. Figure 37 shows the DMA request going away too late to prevent the immediate generation of another DMA transfer. Any wait states inserted in the deposit cycle of the DMA transfer will

lengthen the amount of time from the beginning of the deposit cycle to the time DMA will be sampled for another DMA transfer. Thus, if the amount of time a device requires to drop its DMA request after receiving a DMA acknowledge from the 80186 is longer than the 0 wait state 80186 maximum (100 ns), wait states can be inserted into the DMA cycle to lengthen the amount of time the device has to drop its DMA request after receiving the DMA acknowledge. Table 4 shows the amount of time between the beginning of T_2 and the time DMA request is sampled as wait states are inserted in the DMA deposit cycle.

Table 4. DMA Request Inactive Timing

Number of Wait States	Max Time(ns) For DRQ Inactive From Start of T_2
0	100
1	225
2	350
3	475

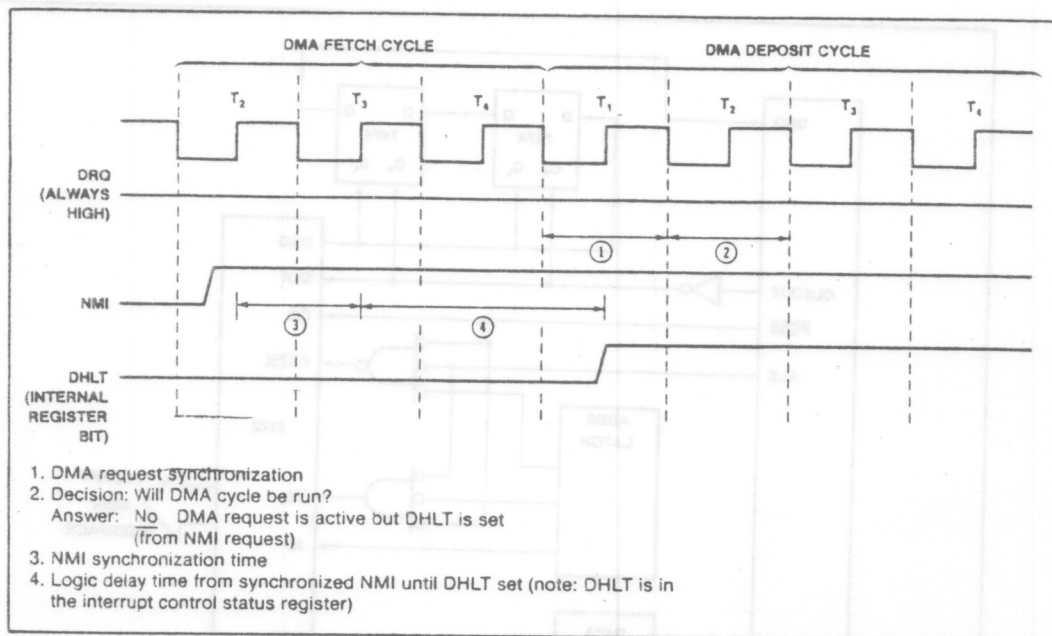


Figure 38. NMI and DMA Interaction

4.8 DMA Halt and NMI

Whenever a Non-Maskable Interrupt is received by the 80186, all DMA activity will be suspended after the end of the current DMA transfer. This is performed by the NMI automatically setting the DMA Halt (DHLT) bit in the interrupt controller status register (see section 7.3.1). The timing of NMI required to prevent a DMA cycle from occurring is shown in Figure 38. After the NMI has been serviced, the DHLT bit should be cleared by the programmer, and DMA activity will resume exactly where it left off, i.e., none of the DMA registers will have been modified. The DMA Halt bit is not automatically reset after the NMI has been serviced. It is automatically reset by the IRET instruction. This DMA halt bit may also be set by the programmer to prevent DMA activity during any critical section of code.

4.9 Example DMA Interfaces

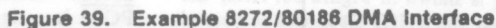
4.9.1 8272 FLOPPY DISK INTERFACE

An example DMA Interface to the 8272 Floppy Disk Controller is shown in Figure 39. This shows how a typical DMA device can be interfaced to the 80186. An example floppy disk software driver for this interface is given in Appendix C.

The data lines of the 8272 are connected, through buffers, to the 80186 AD0-AD7 lines. The buffers are required because the 8272 will not float its output drivers quickly enough to prevent contention with the 80186 driven address information after a read from the 8272 (see section 3.1.3).

DMA acknowledge for the 8272 is driven by an address decode within the region assigned to PCS2. If PCS2 is assigned to be active between I/O locations 0500H and 057FH, then an access to I/O location 0500H will enable only the chip select, while an access to I/O location 0510H will enable both the chip select and the DMA acknowledge. Remember, ALE must be factored into the DACK generation logic because addresses are not guaranteed stable when the chip selects become active. If ALE were not used, the DACK generation circuitry could glitch as address output changed state while the chip select was active.

Notice that the TC line of the 8272 is driven by a very similar circuit as the one generating DACK (except for the reversed sense of the output!). This line is used to terminate an 8272 command before the command has completed execution. Thus, the TC input to the 8272 is software driven in this case. Another method of driving the TC input would be to connect the DACK signal to one of the 80186 timers, and program the timer to out-



DMA requests are delayed by two clock periods in going from the 8272 to the 80186. This is required by the 8272 t_{RQR} (time from DMA request to DMA $\overline{RD}$ going active) spec of 800ns min. This requires 6.4 80186 CPU

DMA request will go away 200ns after DACK is presented to the 8272. During a DMA write cycle (i.e., destination synchronized transfer), this is not soon enough to prevent the immediate generation of another DMA transfer if no wait states are inserted in the deposit cycle to the 8272. Therefore, at least 1 wait state is required by this interface, regardless of the data access parameters of the 8272.

4.9.2 8274 SERIAL COMMUNICATION INTERFACE

An example 8274 synchronous/asynchronous serial chip/80186 DMA interface is shown in Figure 40. The 8274 interface is even simpler than the 8272 interface, since it does not require the generation of a DMA acknowledge signal, and the 8274 does not require the length of time between a DMA request and the DMA read or write cycle that the 8272 does. An example serial driver using the 8274 in DMA mode with the 80186 is given in Appendix C.

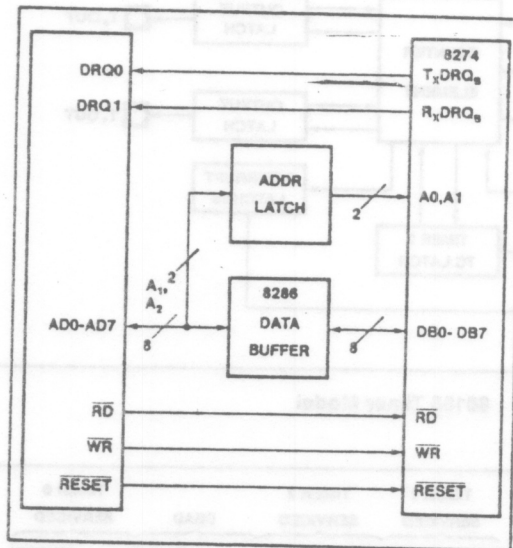


Figure 40. Example 8274/80186 DMA Interface

The data lines of the 8274 are connected through buffers to the 80186 AD0-AD7 lines. Again, these are required not because of bus drive problems, but because the 8274 will not float its drivers before the 80186 will begin driving address information on its address/data bus. If both the 8274 and the 8272 are included in the same 80186 system, they could share the same data bus buffer (as could any other peripheral devices in the system).

The 8274 does not require a DMA acknowledge signal. The first read from or write to the data register of the 8274 after the 8274 generates the DMA request signal will clear the DMA request. The time between when the control signal (RD or WR) becomes active and when the 8274 will drop its DMA request during a DMA write is 150ns, which will require at least one wait state be inserted into the DMA write cycle for proper operation of the interface.

5. TIMER UNIT INTERFACING

The 80186 includes a timer unit which provides three independent 16-bit timers. These timers operate independently of the CPU. Two of these have input and output pins allowing counting of external events and generation of arbitrary waveforms. The third timer can be used as a timer, as a prescaler for the other two timers, or as a DMA request source.

5.1 Timer Operation

The internal timer unit on the 80186 could be modeled by a single counter element, time multiplexed to three register banks, each of which contains different control and count values. These register banks are, in turn, dual ported between the counter element and the 80186 CPU (see Figure 41). Figure 42 shows the timer element sequencing, and the subsequent constraints on input and output signals. If the CPU modifies one of the timer registers, this change will affect the counter element the next time that register is presented to the counter element. There is no connection between the sequencing of the counter element through the timer register banks and the Bus Interface Unit's sequencing through T-states. Timer operation and bus interface operation are completely asynchronous.

5.2 Timer Registers

Each timer is controlled by a block of registers (see Figure 43). Each of these registers can be read or written whether or not the timer is operating. All processor accesses to these registers are synchronized to all counter element accesses to these registers, meaning that one will never read a count register in which only half of the bits have been modified. Because of this synchronization, one wait state is automatically inserted into any access to the timer registers. Unlike the DMA unit, locking accesses to timer registers will not prevent the timer's counter element from accessing the timer registers.

Each timer has a 16-bit count register. This register is incremented for each timer event. A timer event can be a low-to-high transition on the external pin (for timers 0 and 1), a CPU clock transition (divided by 4 because of the counter element multiplexing), or a time out of timer 2 (for timers 0 and 1). Because the count register is 16 bits wide, up to 65536 (2^{16}) timer events can be counted by a single timer/counter. This register can be both read or written whether the timer is or is not operating.

Each timer includes a maximum count register. Whenever the timer count register is equal to the maximum count register, the count register will be reset to zero, that is, the maximum count value will never be stored in the count register. This maximum count value may be written while the timer is operating. A maximum count

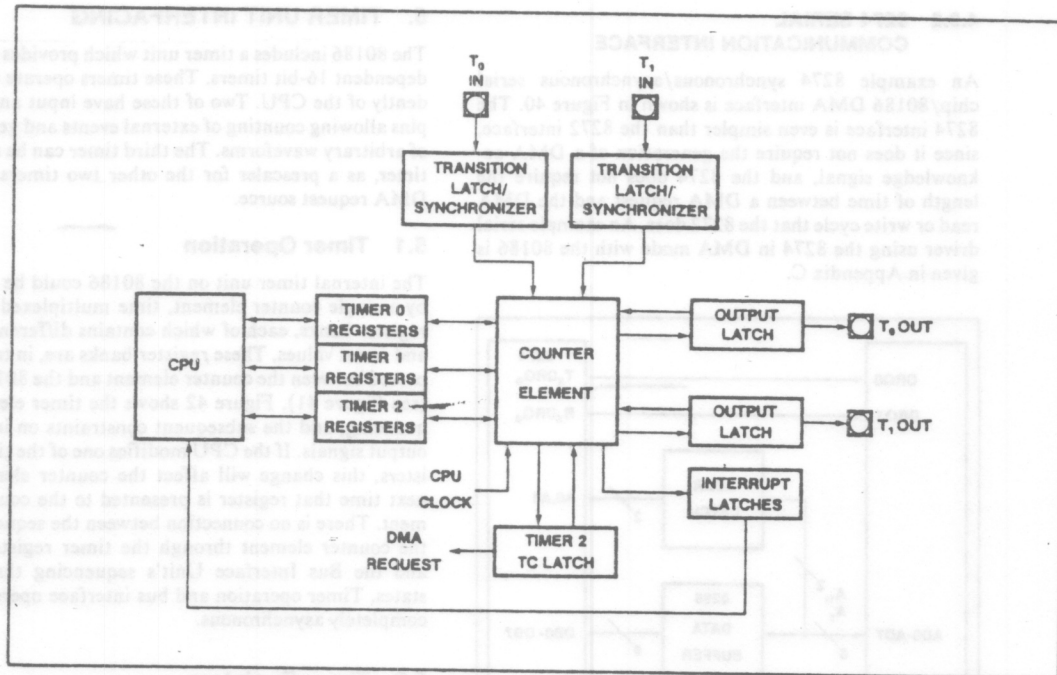


Figure 41. 80186 Timer Model

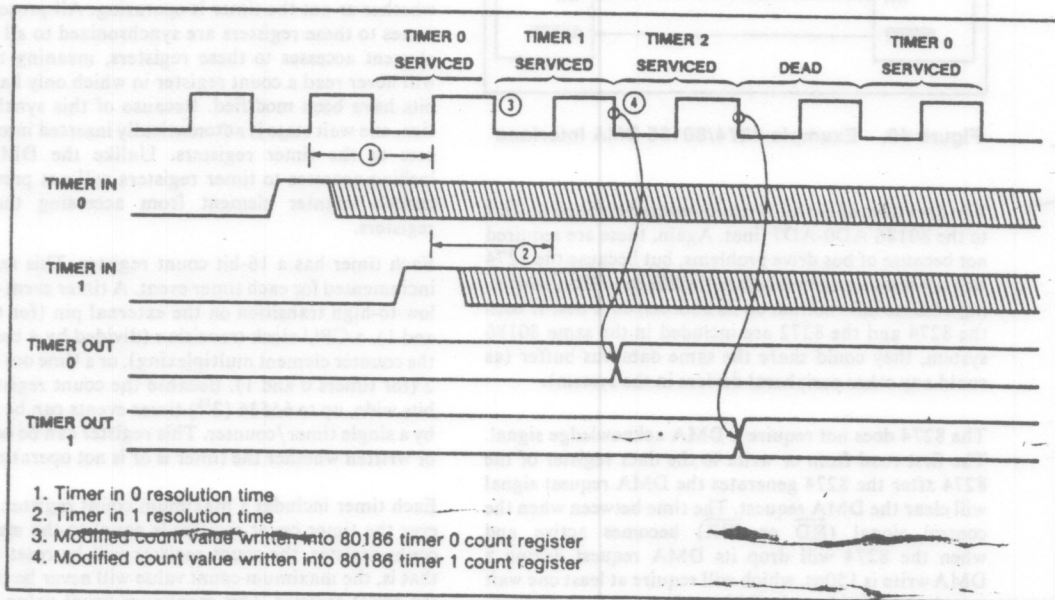


Figure 42. 80186 Counter Element Multiplexing and Timer Input Synchronization

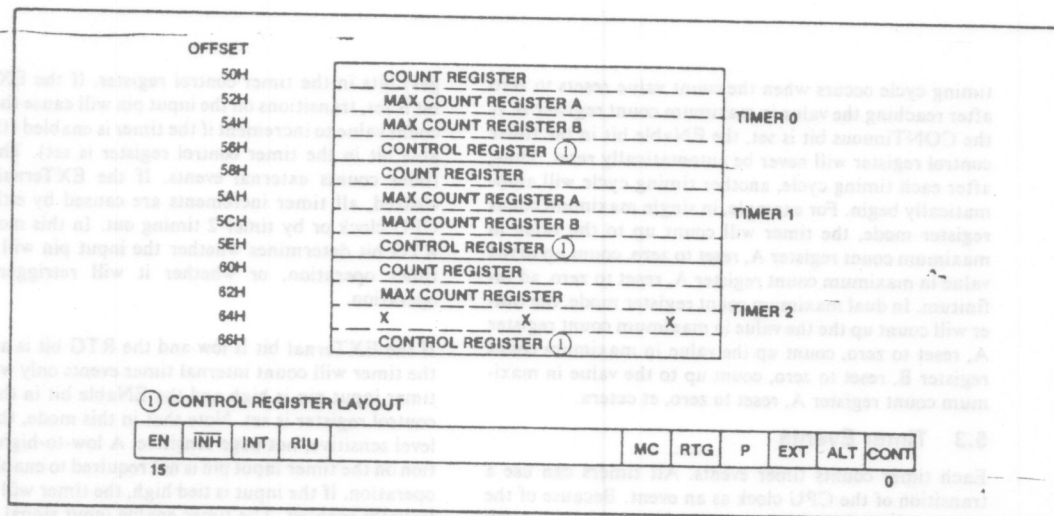


Figure 43. 80186 Timer Register Layout

value of 0 implies a maximum count of 65536, a maximum count value of 1 implies a maximum count of 1, etc. The user should be aware that only equivalence between the count value and the maximum count register value is checked, that is, the count value will not be cleared if the value in the count register is greater than the value in the maximum count register. This could only occur by programmer intervention, either by setting the value in the count register greater than the value in the maximum count register, or by setting the value in the maximum count register to be less than the value in the count register. If this is programmed, the timer will count to the maximum possible count (FFFFH), increment to 0, then count up to the value in the maximum count register. The TC bit in the timer control register will not be set when the counter overflows to 0, nor will an interrupt be generated from the timer unit.

Timers 0 and 1 each contain an additional maximum count register. When both maximum count registers are used, the timer will first count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, and reset to zero again. The ALTerNate bit in the timer control register determines whether one or both maximum count registers are used. If this bit is low, only maximum count register A is used; maximum count register B is ignored. If it is high, both maximum count register A and maximum count register B are used. The RIU (register in use) bit in the timer control register indicates which maximum count register is currently being used. This bit is 0 when maximum count register A is being used, 1 when maximum count register B is being used. This RIU bit is read only. It is unaffected by any write to the timer control register. It will always be read 0 in single maximum count regis-

ter mode (since only maximum count register A will be used).

Each timer can generate an interrupt whenever the timer count value reaches a maximum count value. That is, an interrupt can be generated whenever the value in maximum count register A is reached, and whenever the value in maximum count register B is reached. In addition, the MC (maximum count) bit in the timer control register is set whenever the timer count reaches a maximum count value. This bit is never automatically cleared, i.e., programmer intervention is required to clear this bit. If a timer generates a second interrupt request before the first interrupt request has been serviced, the first interrupt request to the CPU will be lost.

Each timer has an ENable bit in the timer control register. This bit is used to enable the timer to count. The timer will count timer events only when this bit is set. Any timer events occurring when this bit is reset are ignored. Any write to the timer control register will modify the ENable bit only if the INHibit bit is also set. The timer ENable bit will not be modified by a write to the timer control register if the INHibit bit is not set. The INHibit bit in the timer control register allows selective updating of the timer ENable bit. The value of the INHibit bit is not stored in a write to the timer control register; it will always be read as a 1.

Each timer has a CONTinuous bit in the timer control register. If this bit is cleared, the timer ENable bit will be automatically cleared at the end of each timing cycle. If a single maximum count register is used, the end of a timing cycle occurs when the count value resets to zero after reaching the value in maximum count register A. If dual maximum count registers are used, the end of a

timing cycle occurs when the count value resets to zero after reaching the value in maximum count register B. If the CONTinuous bit is set, the ENable bit in the timer control register will never be automatically reset. Thus, after each timing cycle, another timing cycle will automatically begin. For example, in single maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, and increment. In dual maximum count register mode, the timer will count up to the value in maximum count register A, reset to zero, count up to the value in maximum count register B, reset to zero, count up to the value in maximum count register A, reset to zero, et cetera.

5.3 Timer Events

Each timer counts timer events. All timers can use a transition of the CPU clock as an event. Because of the counter element multiplexing, the timer count value will be incremented every fourth CPU clock. For timer 2, this is the only timer event which can be used. For timers 0 and 1, this event is selected by clearing the EXTERNAL and Prescaler bits in the timer control register.

Timers 0 and 1 can use timer 2 reaching its maximum count as a timer event. This is selected by clearing the EXTERNAL bit and setting the Prescaler bit in the timer control register. When this is done, the timer will increment whenever timer 2 resets to zero having reached its own maximum count. Note that timer 2 must be initialized and running for the other timer's value to be incremented.

Timers 0 and 1 can also be programmed to count low-to-high transitions on the external input pin. Each transition on the external pin is synchronized to the 80186 clock before it is presented to the timer circuitry, and may, therefore, be asynchronous (see Appendix B for information on 80186 synchronizers). The timer counts transitions on the input pin: the input value must go low, then go high to cause the timer to increment. Any transition on this line is latched. If a transition occurs when a timer is not being serviced by the counter element, the transition on the input line will be remembered so that when the timer does get serviced, the input transition will be counted. Because of the counter element multiplexing, the maximum rate at which the timer can count is 1/4 of the CPU clock rate (2 MHz with an 8 MHz CPU clock).

5.4 Timer Input Pin Operation

Timers 0 and 1 each have individual timer input pins. All low-to-high transitions on these input pins are synchronized, latched, and presented to the counter element when the particular timer is being serviced by the counter element.

Signals on this input can affect timer operation in three different ways. The manner in which the pin signals are used is determined by the EXTERNAL and RTG (retrigger)

ger) bits in the timer control register. If the EXTERNAL bit is set, transitions on the input pin will cause the timer count value to increment if the timer is enabled (the ENable bit in the timer control register is set). Thus, the timer counts external events. If the EXTERNAL bit is cleared, all timer increments are caused by either the CPU clock or by timer 2 timing out. In this mode, the RTG bit determines whether the input pin will enable timer operation, or whether it will retrigger timer operation.

If the EXTERNAL bit is low and the RTG bit is also low, the timer will count internal timer events only when the timer input pin is high and the ENable bit in the timer control register is set. Note that in this mode, the pin is level sensitive, not edge sensitive. A low-to-high transition on the timer input pin is not required to enable timer operation. If the input is tied high, the timer will be continually enabled. The timer enable input signal is completely independent of the ENable bit in the timer control register: both must be high for the timer to count. Example uses for the timer in this mode would be a real time clock or a baud rate generator.

If the EXTERNAL bit is low and the RTG bit is high, the timer will act as a digital one-shot. In this mode, every low-to-high transition on the timer input pin will cause the timer to reset to zero. If the timer is enabled (i.e., the ENable bit in the timer control register is set) timer operation will begin (the timer will count CPU clock transitions or timer 2 timeouts). Timer operation will cease at the end of a timer cycle, that is, when the value in the maximum count register A is reached and the timer count value resets to zero (in single maximum count register mode, remember that the maximum count value is never stored in the timer count register) or when the value in maximum count register B is reached and the timer count value resets to zero (in dual maximum count register mode). If another low-to-high transition occurs on the input pin before the end of the timer cycle, the timer will reset to zero and begin the timing cycle again regardless of the state of the CONTinuous bit in the timer control register the RIU bit will not be changed by the input transition. If the CONTinuous bit in the timer control register is cleared, the timer ENable bit will automatically be cleared at the end of the timer cycle. This means that any additional transitions on the input pin will be ignored by the timer. If the CONTinuous bit in the timer control register is set, the timer will reset to zero and begin another timing cycle for every low-to-high transition on the input pin, regardless of whether the timer had reached the end of a timer cycle, because the timer ENable bit would not have been cleared at the end of the timing cycle. The timer will also continue counting at the end of a timer cycle, whether or not another transition has occurred on the input pin. An example use of the timer in this mode is an alarm clock time out signal or interrupt.

5.5 Timer Output Pin Operation

Timers 0 and 1 each contain a single timer output pin. This pin can perform two functions at programmer option. The first is a single pulse indicating the end of a timing cycle. The second is a level indicating the maximum count register currently being used. The timer outputs operate as outlined below whether internal or external clocking of the timer is used. If external clocking is used, however, the user should remember that the time between an external transition on the timer input pin and the time this transition is reflected in the timer output pin will vary depending on when the input transition occurs relative to the timer's being serviced by the counter element.

When the timer is in single maximum count register mode (the ALTERNATE bit in the timer control register is cleared) the timer output pin will go low for a single CPU clock the clock after the timer is serviced by the counter element where maximum count is reached (see Figure 44). This mode is useful when using the timer as

a baud rate generator.

When the timer is programmed in dual maximum count register mode (the ALTERNATE bit in the timer control register is set), the timer output pin indicates which maximum count register is being used. It is low if maximum count register B is being used for the current count, high if maximum count register A is being used. If the timer is programmed in continuous mode (the CONTINUOUS bit in the timer control register is set), this pin could generate a waveform of any duty cycle. For example, if maximum count register A contained 10 and maximum count register B contained 20, a 33% duty cycle waveform would be generated.

5.6 Sample 80186 Timer Applications

The 80186 timers can be used for almost any application for which a discrete timer circuit would be used. These include real time clocks, baud rate generators, or event counters.

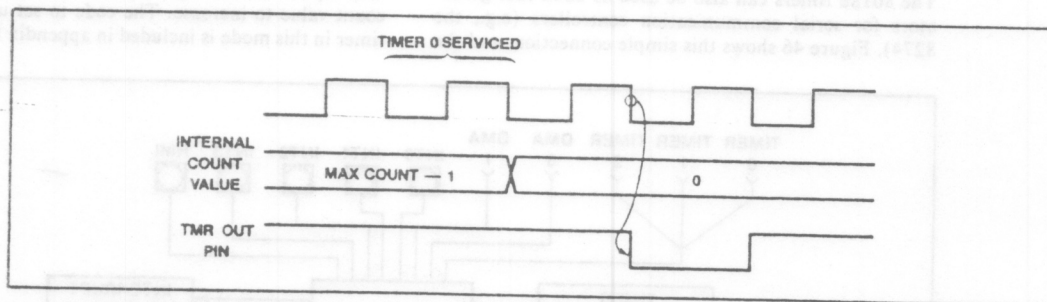


Figure 44. 80186 Timer Out Signal

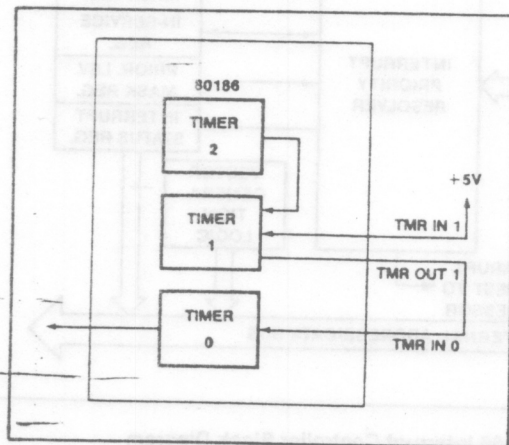


Figure 45. 80186 Real Time Clock

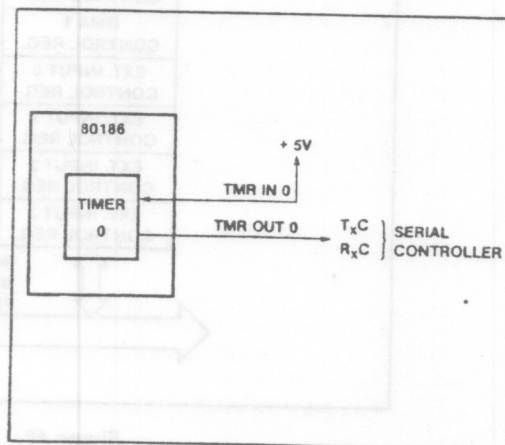


Figure 46. 80186 Baud Rate Generator

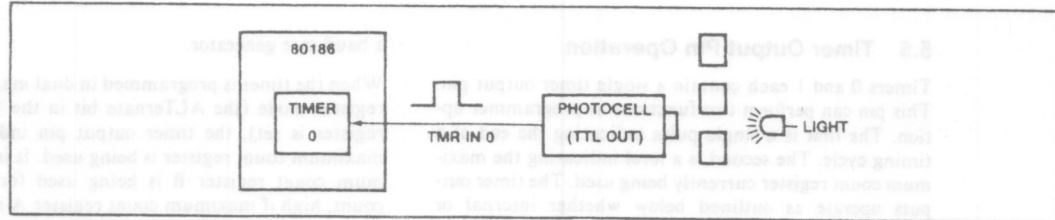


Figure 47.

5.6.1 80186 TIMER REAL TIME CLOCK

The sample program in appendix D shows the 80186 timer being used with the 80186 CPU to form a real time clock. In this implementation, timer 2 is programmed to provide an interrupt to the CPU every millisecond. The CPU then increments memory based clock variables.

5.6.2 80186 TIMER BAUD RATE GENERATOR

The 80186 timers can also be used as baud rate generators for serial communication controllers (e.g., the 8274). Figure 46 shows this simple connection, and the

code to program the timer as a baud rate generator is included in appendix D.

5.6.3 80186 TIMER EVENT COUNTER

The 80186 timer can be used to count events. Figure 47 shows a hypothetical set up in which the 80186 timer will count the interruptions in a light source. The number of interruptions can be read directly from the count register of the timer, since the timer counts up, i.e., each interruption in the light source will cause the timer count value to increase. The code to set up the 80186 timer in this mode is included in appendix D.

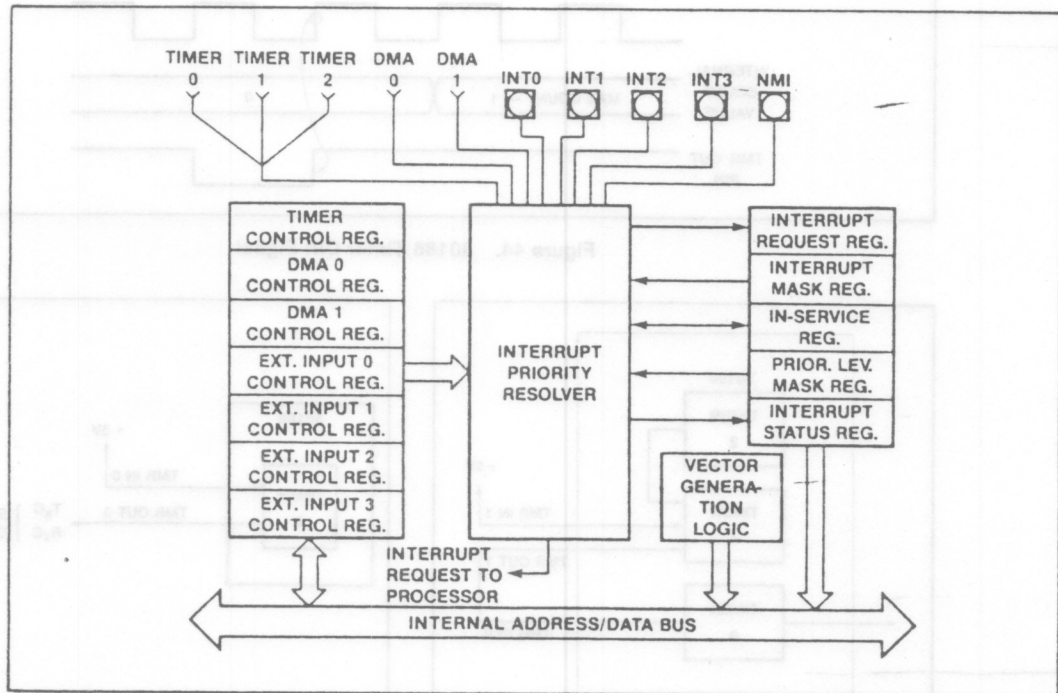


Figure 48. 80186 Interrupt Controller Block Diagram

6. 80186 INTERRUPT CONTROLLER INTERFACING

The 80186 contains an integrated interrupt controller. This unit performs tasks of the interrupt controller in a typical system. These include synchronization of interrupt requests, prioritization of interrupt requests, and request type vectoring in response to a CPU interrupt acknowledge. It can be a master to two external 8259A interrupt controllers or can be a slave to an external interrupt controller to allow compatibility with the iRMX 86 operating system, and the 80130/80150 operating system firmware chips.

6.1 Interrupt Controller Model

The integrated interrupt controller block diagram is shown in Figure 48. It contains registers and a control element. Four inputs are provided for external interfacing to the interrupt controller. Their functions change according to the programmed mode of the interrupt controller. Like the other 80186 integrated peripheral registers, the interrupt controller registers are available for CPU reading or writing at any time.

6.2 Interrupt Controller Operation

The interrupt controller operates in two major modes, non-iRMX 86 mode (referred to henceforth as **master mode**), and **iRMX 86 mode**. In master mode the integrated controller acts as the master interrupt controller for the system, while in iRMX 86 mode the controller

operates as a slave to an external interrupt controller which operates as the master interrupt controller for the system. Some of the interrupt controller registers and interrupt controller pins change definition between these two modes, but the basic charter and function of the interrupt controller remains fundamentally the same. The difference is when in master mode, the interrupt controller presents its interrupt input directly to the 80186 CPU, while in iRMX 86 mode the interrupt controller presents its interrupt input to an external controller (which then presents its interrupt input to the 80186 CPU). Placing the interrupt controller in iRMX 86 mode is done by setting the iRMX mode bit in the peripheral control block pointer (see appendix A).

6.3 Interrupt Controller Registers

The interrupt controller has a number of registers which are used to control its operation (see Figure 49). Some of these change their function between the two major modes of the interrupt controller (master and iRMX 86 mode). The differences are indicated in the following section. If not indicated, the function and implementation of the registers is the same in the two basic modes of operation of the interrupt controller. The method of interaction among the various interrupt controller registers is shown in the flowcharts in Figures 57 and 58.

6.3.1 CONTROL REGISTERS

Each source of interrupt to the 80186 has a control register in the internal controller. These registers contain

MASTER MODE	OFFSET ADDRESS	iRMX86™ Mode
INT3 CONTROL REGISTER	3EH	①
INT2 CONTROL REGISTER	3CH	①
INT1 CONTROL REGISTER	3AH	TIMER 2 CONTROL REGISTER
INT0 CONTROL REGISTER	38H	TIMER 1 CONTROL REGISTER
DMA1 CONTROL REGISTER	36H	DMA1 CONTROL REGISTER
DMA0 CONTROL REGISTER	34H	DMA0 CONTROL REGISTER
TIMER CONTROL REGISTER	32H	TIMER 0 CONTROL REGISTER
INTERRUPT CONTROLLER STATUS REGISTER	30H	INTERRUPT CONTROLLER STATUS REGISTER
INTERRUPT REQUEST REGISTER	2EH	INTERRUPT REQUEST REGISTER
IN-SERVICE REGISTER	2CH	IN SERVICE REGISTER
PRIORITY MASK REGISTER	2AH	PRIORITY MASK REGISTER
MASK REGISTER	28H	MASK REGISTER
POLL STATUS REGISTER	26H	①
POLL REGISTER	24H	①
EOI REGISTER	22H	SPECIFIC EOI REGISTER
①	20H	INTERRUPT VECTOR REGISTER

1. Unsupported in this mode: values written may or may not be stored

Figure 49. 80186 Interrupt Controller Registers

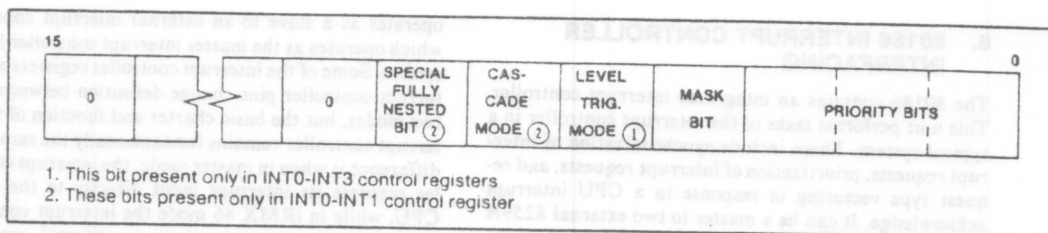


Figure 50. Interrupt Controller Control Register

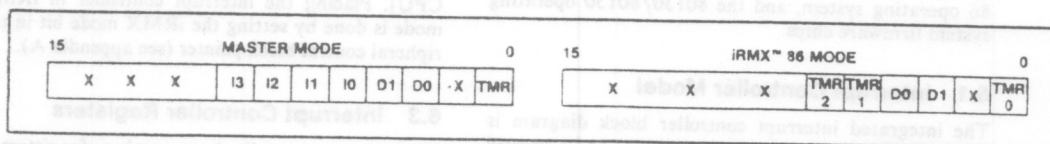


Figure 51. 80186 Interrupt Controller in-Service, Interrupt Request and Mask Register Format

three bits which select one of eight different interrupt priority levels for the interrupt device (0 is highest priority, 7 is lowest priority), and a mask bit to enable the interrupt (see Figure 50). When the mask bit is low, the interrupt is enabled, when it is high, the interrupt is masked.

There are seven control registers in the 80186 integrated interrupt controller. In master mode, four of these serve the external interrupt inputs, one each for the two DMA channels, and one for the collective timer interrupts. In iRMX 86 mode, the external interrupt inputs are not used, so each timer can have its own individual control register.

6.3.2 REQUEST REGISTER

The interrupt controller includes an interrupt request register (see Figure 51). This register contains seven active bits, one for each interrupt control register. Whenever an interrupt request is made by the interrupt source associated with a specific control register, the bit in interrupt request register is set, regardless if the interrupt is enabled, or if it is of sufficient priority to cause a processor interrupt. The bits in this register which are associated with integrated peripheral devices (the DMA and timer units) can be read or written, while the bits in this register which are associated with the external interrupt pins can only be read (values written to them are not stored). These interrupt request bits are automatically cleared when the interrupt is acknowledged.

6.3.3 MASK REGISTER AND PRIORITY MASK REGISTER

The interrupt controller contains a mask register (see Figure 51). This register contains a mask bit for each interrupt source associated with an interrupt control register. The bit for an interrupt source in the mask register is

identically the same bit as is provided in the interrupt control register: modifying a mask bit in the control register will also modify it in the mask register, and vice versa.

The interrupt controller also contains a priority mask register (see Figure 52). This register contains three bits which indicate the priority of the current interrupt being serviced. When an interrupt is acknowledged (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register, see below), these bits are set to the priority of the device whose interrupt is being acknowledged (which will never be lower than the previous priority programmed into these bits). They prevent any interrupt of lower priority (as set by the priority bits in the interrupt control registers for interrupt sources) from interrupting the processor. These bits are automatically set to the priority of the next lowest interrupt when the End Of Interrupt is issued by the CPU to the interrupt controller (or all 1's if there is no interrupt pending, meaning that interrupts of all priority levels are enabled). This register may be read or written.

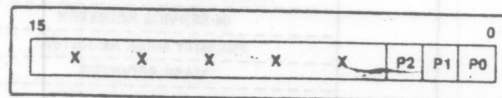


Figure 52. 80186 Interrupt Controller Priority Mask Register Format

6.3.4 IN-SERVICE REGISTER

The interrupt controller contains an in-service register (see Figure 51). A bit in the in-service register is associated with each interrupt control register so that when an interrupt request by the device associated with the con-

rol register is acknowledged by the processor (either by the processor running the interrupt acknowledge or by the processor reading the interrupt poll register) the bit is set. The bit is reset when the CPU issues an End Of Interrupt to the interrupt controller. This register may be both read and written, i.e., the CPU may set in-service bits without an interrupt ever occurring, or may reset them without using the EOI function of the interrupt controller.

5.3.5 POLL AND POLL STATUS REGISTERS

The interrupt controller contains both a poll register and a poll status register (see Figure 53). Both of these registers contain the same information. They have a single bit to indicate an interrupt is pending. This bit is set if an interrupt of sufficient priority has been received. It is automatically cleared when the interrupt is acknowledged. If (and only if) an interrupt is pending, they also contain information as to the interrupt type of the highest priority interrupt pending.

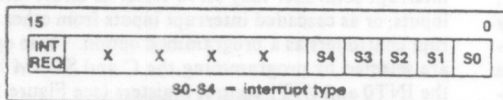


Figure 53. 80186 Poll & Poll Status Register Format

Reading the poll register will acknowledge the pending interrupt to the interrupt controller just as if the processor had acknowledged the interrupt through interrupt acknowledge cycles. The processor will not actually run

any interrupt acknowledge cycles, and will not vector through a location in the interrupt vector table. Only the interrupt request, in-service and priority mask registers in the interrupt controller are set appropriately. Reading the poll status register will merely transmit the status of the polling bits without modifying any of the other interrupt controller registers. These registers are read only: data written to them is not stored. These registers are not supported in iRMX 86 mode. The state of the bits in these registers in iRMX 86 mode is not defined. However, accessing the poll register location when in iRMX 86 mode will cause the interrupt controller to "acknowledge" the interrupt (i.e., the in-service bit and priority level mask register bits will be set).

6.3.6 END OF INTERRUPT REGISTER

The interrupt controller contains an End Of Interrupt register (see Figure 54). The programmer issues an End Of Interrupt to the controller by writing to this register. After receiving the End Of Interrupt, the interrupt controller automatically resets the in-service bit for the interrupt and the priority mask register bits. The value of the word written to this register determines whether the End Of Interrupt is specific or non-specific. A non-specific End Of Interrupt is specified by setting the non-specific bit in the word written to the End Of Interrupt register. In a non-specific End Of Interrupt, the in-service bit of the highest priority interrupt set is automatically cleared, while a specific End Of Interrupt allows the in-service bit cleared to be explicitly specified. The in-service bit is reset whether the bit was set by an interrupt acknowledge or if it was set by the CPU writing the

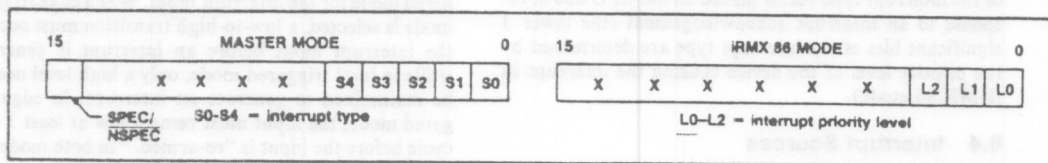


Figure 54. 80186 End of Interrupt Register Format

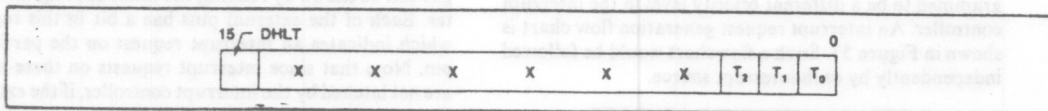


Figure 55. 80186 Interrupt Status Register Format

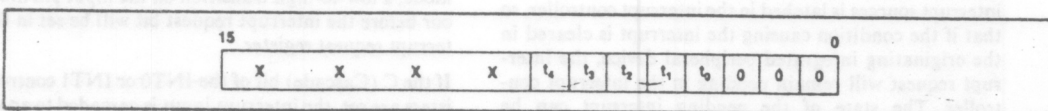


Figure 56. 80186 Interrupt Vector Register Format (iRMX 86 mode only)

bit directly to the in-service register. If the highest priority interrupt is reset, the priority mask register bits will change to reflect the next lowest priority interrupt to be serviced. If a less than highest priority interrupt in-service bit is reset, the priority mask register bits will not be modified (because the highest priority interrupt being serviced has not changed). Only the specific EOI is supported in iRMX 86 mode. This register is write only; data written is not stored and cannot be read back.

6.3.7 INTERRUPT STATUS REGISTER

The interrupt controller also contains an interrupt status register (see Figure 55). This register contains four significant bits. There are three bits used to show which timer is causing an interrupt. This is required because in master mode, the timers share a single interrupt control register. A bit in this register is set to indicate which timer has generated an interrupt. The bit associated with a timer is automatically cleared after the interrupt request for the timer is acknowledged. More than one of these bits may be set at a time. The fourth bit in the interrupt status register is the DMA halt bit. When set, this bit prevents any DMA activity. It is automatically set whenever a NMI is received by the interrupt controller. It can also be set explicitly by the programmer. This bit is automatically cleared whenever the IRET instruction is executed. All significant bits in this register are read/write.

6.3.8 INTERRUPT VECTOR REGISTER

Finally, in iRMX 86 mode only, the interrupt controller contains an interrupt vector register (see Figure 56). This register is used to specify the 5 most significant bits of the interrupt type vector placed on the CPU bus in response to an interrupt acknowledgement (the lower 3 significant bits of the interrupt type are determined by the priority level of the device causing the interrupt in iRMX 86 mode).

6.4 Interrupt Sources

The 80186 interrupt controller receives and arbitrates among many different interrupt request sources, both internal and external. Each interrupt source may be programmed to be a different priority level in the interrupt controller. An interrupt request generation flow chart is shown in Figure 57. Such a flowchart would be followed independently by each interrupt source.

6.4.1 INTERNAL INTERRUPT SOURCES

The internal interrupt sources are the three timers and the two DMA channels. An interrupt from each of these interrupt sources is latched in the interrupt controller, so that if the condition causing the interrupt is cleared in the originating integrated peripheral device, the interrupt request will remain pending in the interrupt controller. The state of the pending interrupt can be obtained by reading the interrupt request register of the

interrupt controller. For all internal interrupts, the latched interrupt request can be reset by the processor by writing to the interrupt request register. Note that all timers share a common bit in the interrupt request register in master mode. The interrupt controller status register may be read to determine which timer is actually causing the interrupt request in this mode. Each timer has a unique interrupt vector (see section 6.5.1). Thus polling is not required to determine which timer has caused the interrupt in the interrupt service routine. Also, because the timers share a common interrupt control register, they are placed at a common priority level as referenced to all other interrupt devices. Among themselves they have a fixed priority, with timer 0 as the highest priority timer and timer 2 as the lowest priority timer.

6.4.2 EXTERNAL INTERRUPT SOURCES

The 80186 interrupt controller will accept external interrupt requests only when it is programmed in master mode. In this mode, the external pins associated with the interrupt controller may serve either as direct interrupt inputs, or as cascaded interrupt inputs from other interrupt controllers as a programmed option. These options are selected by programming the C and SFNM bits in the INT0 and INT1 control registers (see Figure 50).

When programmed as direct interrupt inputs, the four interrupt inputs are each controlled by an individual interrupt control register. As stated earlier, these registers contain 3 bits which select the priority level for the interrupt and a single bit which enables the interrupt source to the processor. In addition each of these control registers contains a bit which selects either edge or level triggered mode for the interrupt input. When edge triggered mode is selected, a low-to-high transition must occur on the interrupt input before an interrupt is generated, while in level triggered mode, only a high level needs to be maintained to generate an interrupt. In edge triggered mode, the input must remain low at least 1 clock cycle before the input is "re-armed." In both modes, the interrupt level must remain high until the interrupt is acknowledged, i.e., the interrupt request is not latched in the interrupt controller. The status of the interrupt input can be shown by reading the interrupt request register. Each of the external pins has a bit in this register which indicates an interrupt request on the particular pin. Note that since interrupt requests on these inputs are not latched by the interrupt controller, if the external input goes inactive, the interrupt request (and also the bit in the interrupt request register) will also go inactive (low). Also, if the interrupt input is in edge triggered mode, a low-to-high transition on the input pin must occur before the interrupt request bit will be set in the interrupt request register.

If the C (Cascade) bit of the INT0 or INT1 control registers are set, the interrupt input is cascaded to an external interrupt controller. In this mode, whenever the

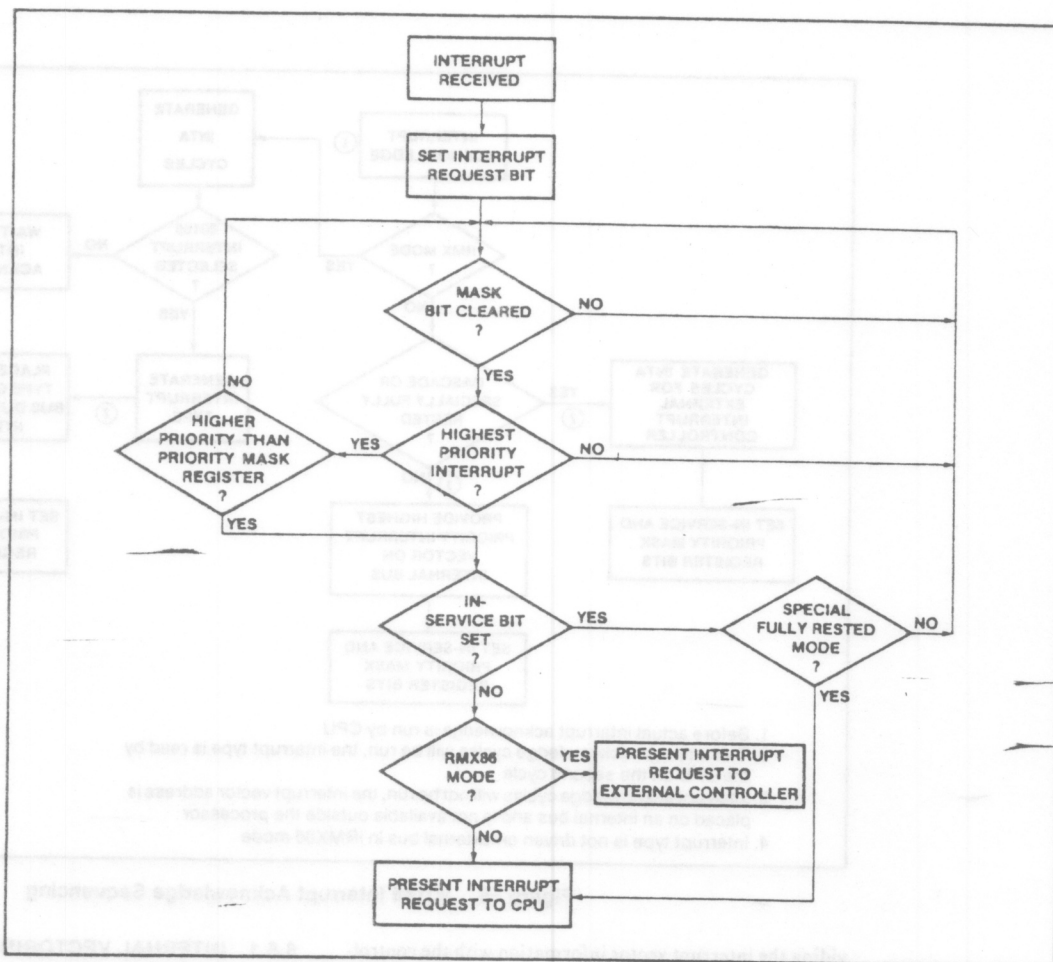


Figure 57. 80186 Interrupt Request Sequencing

interrupt presented to the INT0 or INT1 line is acknowledged, the integrated interrupt controller will not provide the interrupt type for the interrupt. Instead, two INTA bus cycles will be run, with the INT2 and INT3 lines providing the interrupt acknowledge pulses for the INT0 and the INT1 interrupt requests respectively. INT0/INT2 and INT1/INT3 may be individually programmed into cascade mode. This allows 128 individually vectored interrupt sources if two banks of 9 external interrupt controllers each are used.

6.4.3 iRMX™ 86 MODE INTERRUPT SOURCES

When the interrupt controller is configured in iRMX 86 mode, the integrated interrupt controller accepts inter-

rupt requests only from the integrated peripherals. Any external interrupt requests must go through an external interrupt controller. This external interrupt controller requests interrupt service directly from the 80186 CPU through the INT0 line on the 80186. In this mode, the function of this line is not affected by the integrated interrupt controller. In addition, in iRMX 86 mode the integrated interrupt controller must request interrupt service through this external interrupt controller. This interrupt request is made on the INT3 line (see section 6.7.4 on external interrupt connections).

6.5 Interrupt Response

The 80186 can respond to an interrupt in two different ways. The first will occur if the internal controller is pro-

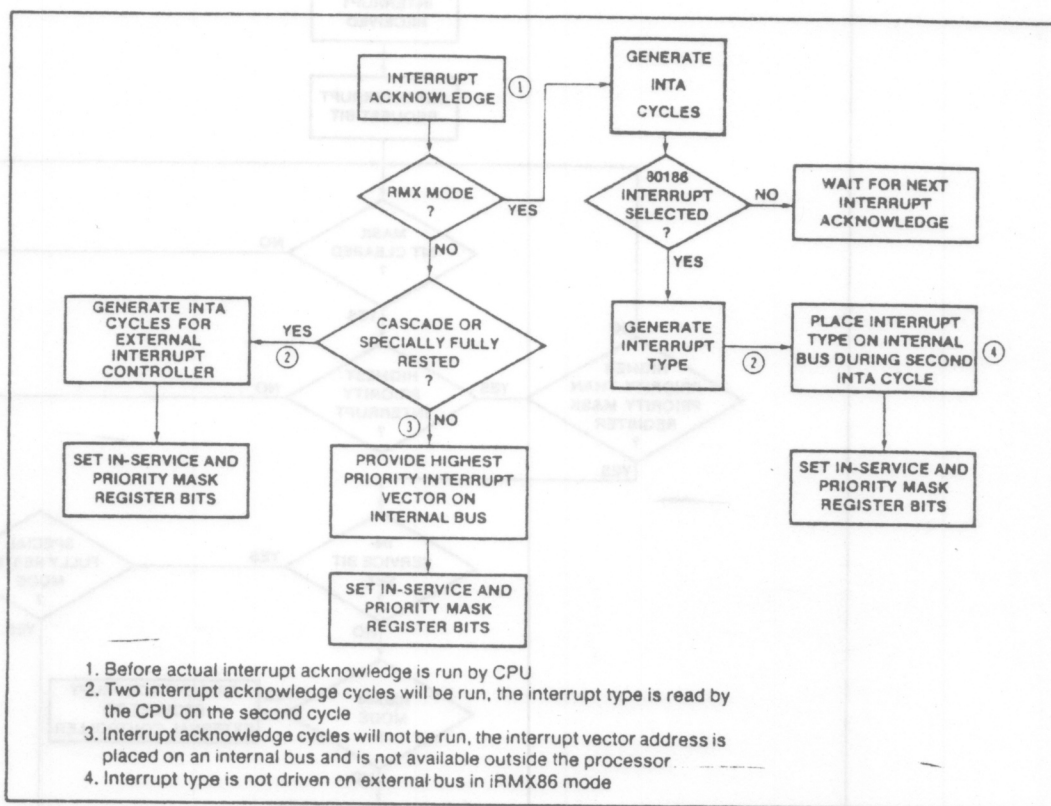


Figure 58. 80186 Interrupt Acknowledge Sequencing

viding the interrupt vector information with the controller in master mode. The second will occur if the CPU reads interrupt type information from an external interrupt controller or if the interrupt controller is in iRMX 86 mode. In both of these instances the interrupt vector information driven by the 80186 integrated interrupt controller is not available outside the 80186 microprocessor.

In each interrupt mode, when the integrated interrupt controller receives an interrupt response, the interrupt controller will automatically set the in-service bit and the priority mask bits and reset the interrupt request bit in the integrated controller. The priority mask bits will prevent the controller from generating any further interrupts to the CPU from sources of lower priority until the higher priority interrupt service routine has run. In addition, unless the interrupt control register for the interrupt is set in Special Fully Nested Mode, the interrupt controller will prevent any interrupts from occurring from the same interrupt line until the in-service bit for that line has been cleared.

6.5.1 INTERNAL VECTORING, MASTER MODE

In master mode, the interrupt types associated with all the interrupt sources are fixed and unalterable. These interrupt types are given in Table 5. In response to an internal CPU interrupt acknowledge the interrupt controller will generate the vector address rather than the interrupt type. On the 80186 (like the 8086) the interrupt vector address is the interrupt type multiplied by 4. This speeds interrupt response.

In master mode, the integrated interrupt controller is the master interrupt controller of the system. As a result, no external interrupt controller need know when the integrated controller is providing an interrupt vector, nor when the interrupt acknowledge is taking place. As a result, no interrupt acknowledge bus cycles will be generated. The first external indication that an interrupt has been acknowledged will be the processor reading the interrupt vector from the interrupt vector table in low memory.

Table 5. 80186 Interrupt Vector Types

Interrupt Name	Vector Type	Default Priority
timer 0	8	0a
timer 1	18	0b
timer 2	19	0c
DMA 0	10	2
DMA 1	11	3
INT 0	12	4
INT 1	13	5
INT 2	14	6
INT 3	15	7

Because the two interrupt acknowledge cycles are not run, and the interrupt vector address does not need to be calculated, interrupt response to an internally vectored interrupt is 42 clock cycles, which is faster than the interrupt response when external vectoring is required, or if the interrupt controller is run in iRMX 86 mode.

If two interrupts of the same programmed priority occur, the default priority scheme (as shown in table 5) is used.

6.5.2 INTERNAL VECTURING, iRMX™ 86 MODE

In iRMX 86 mode, the interrupt types associated with the various interrupt sources are alterable. The upper 5 most significant bits are taken from the interrupt vector register, and the lower 3 significant bits are taken from the priority level of the device causing the interrupt. Because the interrupt type, rather than the interrupt vector address, is given by the interrupt controller in this mode the interrupt vector address must be calculated by the CPU before servicing the interrupt.

In iRMX 86 mode, the integrated interrupt controller will present the interrupt type to the CPU in response to the two interrupt acknowledge bus cycles run by the processor. During the first interrupt acknowledge cycle, the external master interrupt controller determines which slave interrupt controller will be allowed to place its interrupt vector on the microprocessor bus. During the second interrupt acknowledge cycle, the processor reads the interrupt vector from its bus. Thus, these two interrupt acknowledge cycles must be run, since the integrated controller will present the interrupt type information only when the external interrupt controller signals the integrated controller that it has the highest pending interrupt request (see Figure 59). The 80186 samples the

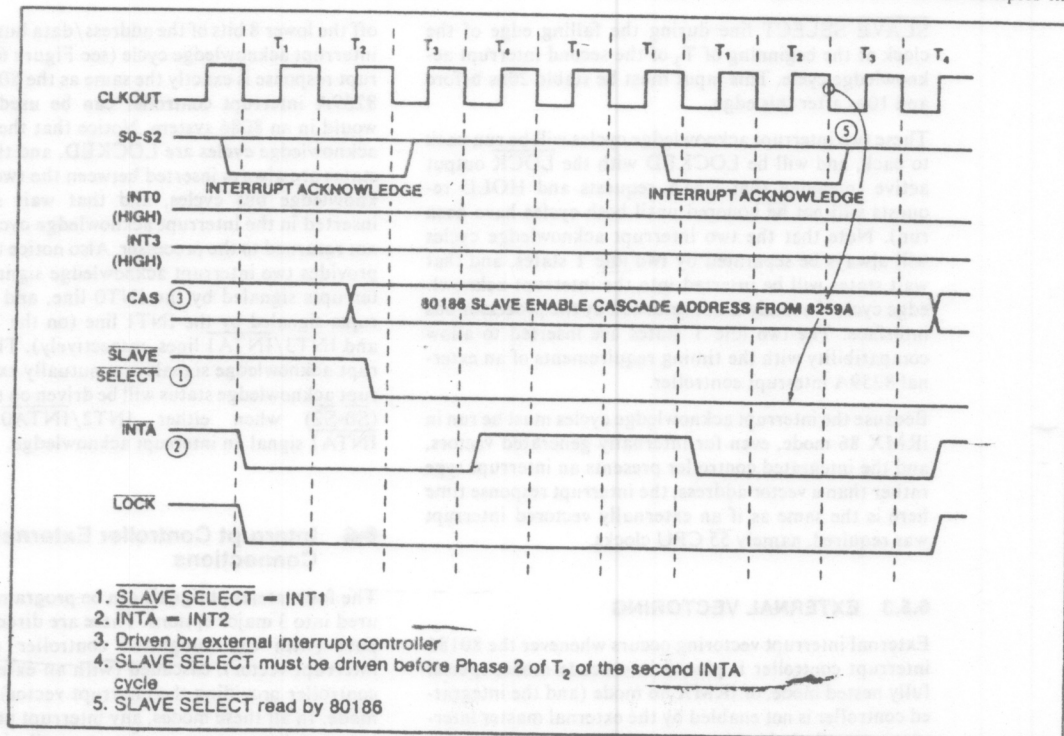


Figure 59. 80186 iRMX-86 Mode Interrupt Acknowledge Timing

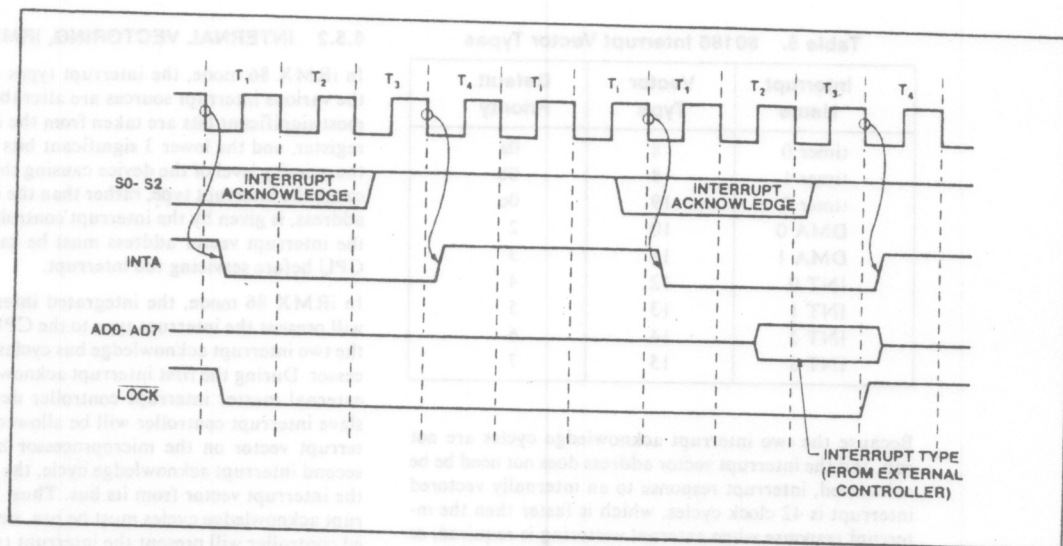


Figure 60. 80186 Cascaded Interrupt Acknowledge Timing

SLAVE SELECT line during the falling edge of the clock at the beginning of T_3 of the second interrupt acknowledge cycle. This input must be stable 20ns before and 10ns after this edge.

These two interrupt acknowledge cycles will be run back to back, and will be LOCKED with the LOCK output active (meaning that DMA requests and HOLD requests will not be honored until both cycles have been run). Note that the two interrupt acknowledge cycles will always be separated by two idle T states, and that wait states will be inserted into the interrupt acknowledge cycle if a ready is not returned by the processor bus interface. The two idle T states are inserted to allow compatibility with the timing requirements of an external 8259A interrupt controller.

Because the interrupt acknowledge cycles must be run in iRMX 86 mode, even for internally generated vectors, and the integrated controller presents an interrupt type rather than a vector address, the interrupt response time here is the same as if an externally vectored interrupt was required, namely 35 CPU clocks.

6.5.3 EXTERNAL VECTORING

External interrupt vectoring occurs whenever the 80186 interrupt controller is placed in cascade mode, special fully nested mode, or iRMX 86 mode (and the integrated controller is not enabled by the external master interrupt controller). In this mode, the 80186 generates two interrupt acknowledge cycles, reading the interrupt type

off the lower 8 bits of the address/data bus on the second interrupt acknowledge cycle (see Figure 60). This interrupt response is exactly the same as the 8086, so that the 8259A interrupt controller can be used exactly as it would in an 8086 system. Notice that the two interrupt acknowledge cycles are LOCKED, and that two idle T-states are always inserted between the two interrupt acknowledge bus cycles, and that wait states will be inserted in the interrupt acknowledge cycle if a ready is not returned to the processor. Also notice that the 80186 provides two interrupt acknowledge signals, one for interrupts signaled by the INT0 line, and one for interrupts signaled by the INT1 line (on the INT2/INTA0 and INT3/INTA1 lines, respectively). These two interrupt acknowledge signals are mutually exclusive. Interrupt acknowledge status will be driven on the status lines (S0-S2) when either INT2/INTA0 or INT3/INTA1 signal an interrupt acknowledge.

6.6 Interrupt Controller External Connections

The four interrupt signals can be programmably configured into 3 major options. These are direct interrupt inputs (with the integrated controller providing the interrupt vector), cascaded (with an external interrupt controller providing the interrupt vector), or iRMX 86 mode. In all these modes, any interrupt presented to the external lines must remain set until the interrupt is acknowledged.

6.6.1 DIRECT INPUT MODE

When the Cascade mode bits are cleared, the interrupt input lines are configured as direct interrupt input lines (see Figure 61). In this mode an interrupt source (e.g., an 8272 floppy disk controller) may be directly connected to the interrupt input line. Whenever an interrupt is received on the input line, the integrated controller will do nothing unless the interrupt is enabled, and it is the highest priority pending interrupt. At this time, the interrupt controller will present the interrupt to the CPU and wait for an interrupt acknowledge. When the acknowledge occurs, it will present the interrupt vector address to the CPU. In this mode, the CPU will not run any interrupt acknowledge cycles.

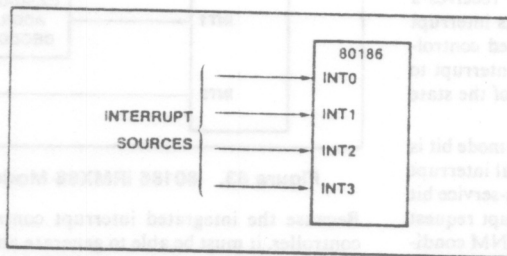


Figure 61. 80186 Non-Cascaded Interrupt Connection

These lines can be individually programmed in either edge or level triggered mode using their respective control registers. In edge triggered mode, a low-to-high transition must occur before the interrupt will be generated to the CPU, while in level triggered mode, only a high level must be present on the input for an interrupt to be generated. In edge trigger mode, the interrupt input must also be low for at least 1 CPU clock cycle to insure recognition. In both modes, the interrupt input must remain active until acknowledged.

6.6.2 CASCADE MODE

When the Cascade mode bit is set and the SFNM bit is cleared, the interrupt input lines are configured in cascade mode. In this mode, the interrupt input line is paired with an interrupt acknowledge line. The INT2/INTA0 and INT3/INTA1 lines are dual purpose; they can function as direct input lines, or they can function as interrupt acknowledge outputs. INT2/INTA0 provides the interrupt acknowledge for an INT0 input, and INT3/INTA1 provides the interrupt acknowledge for an INT1 input. Figure 62 shows this connection.

When programmed in this mode, in response to an interrupt request on the INT0 line, the 80186 will provide two interrupt acknowledge pulses. These pulses will be provided on the INT2/INTA0 line, and will also be reflected by interrupt acknowledge status being generated

on the $\overline{S0}$ - $\overline{S2}$ status lines. On the second pulse, the interrupt type will be read in. The 80186 externally vectored interrupt response is covered in more detail in section 6.5.

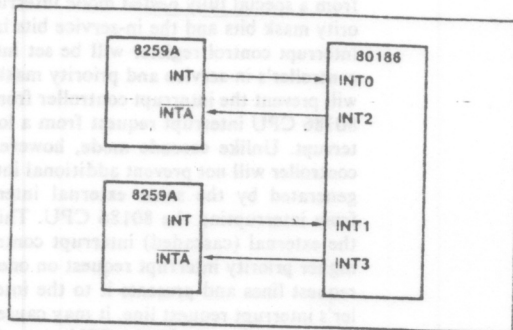


Figure 62. 80186 Cascade and Special Fully Nested Mode Interface

INT0/INT2/INTA0 and INT1/INT3/INTA1 may be individually programmed into interrupt request/acknowledge pairs, or programmed as direct inputs. This means that INT0/INT2/INTA0 may be programmed as an interrupt/acknowledge pair, while INT1 and INT3/INTA1 each provide separate internally vectored interrupt inputs.

When an interrupt is received on a cascaded interrupt, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's mask and priority mask registers. This will prevent the controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Also, since the in-service bit is set, any subsequent interrupt requests on the particular interrupt input line will not cause the integrated interrupt controller to generate an interrupt request to the 80186 CPU. This means that if the external interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the 80186 interrupt request line, it will not subsequently be presented to the 80186 CPU by the integrated interrupt controller until the in-service bit for the interrupt line has been cleared.

6.6.3 SPECIAL FULLY NESTED MODE

When both the Cascade mode bit and the SFNM bit are set, the interrupt input lines are configured in Special Fully Nested Mode. The external interface in this mode is exactly as in Cascade Mode. The only difference is in the conditions allowing an interrupt from the external interrupt controller to the integrated interrupt controller to interrupt the 80186 CPU.

When an interrupt is received from a special fully nested

mode interrupt line, it will interrupt the 80186 CPU if it is the highest priority interrupt pending regardless of the state of the in-service bit for the interrupt source in the interrupt controller. When an interrupt is acknowledged from a special fully nested mode interrupt line, the priority mask bits and the in-service bits in the particular interrupt control register will be set into the interrupt controller's in-service and priority mask registers. This will prevent the interrupt controller from generating an 80186 CPU interrupt request from a lower priority interrupt. Unlike cascade mode, however, the interrupt controller will not prevent additional interrupt requests generated by the same external interrupt controller from interrupting the 80186 CPU. This means that if the external (cascaded) interrupt controller receives a higher priority interrupt request on one of its interrupt request lines and presents it to the integrated controller's interrupt request line, it may cause an interrupt to be generated to the 80186 CPU, regardless of the state of the in-service bit for the interrupt line.

If the SFNM mode bit is set and the Cascade mode bit is not also set, the controller will provide internal interrupt vectoring. It will also ignore the state of the in-service bit in determining whether to present an interrupt request to the CPU. In other words, it will use the SFNM conditions of interrupt generation with an internally vectored interrupt response, i.e., if the interrupt pending is the highest priority type pending, it will cause a CPU interrupt regardless of the state of the in-service bit for the interrupt.

6.6.4 iRMX™ 86 MODE

When the RMX bit in the peripheral relocation register is set, the interrupt controller is set into iRMX 86 mode.

In this mode, all four interrupt controller input lines are used to perform the necessary handshaking with the external master interrupt controller. Figure 63 shows the hardware configuration of the 80186 interrupt lines with an external controller in iRMX 86 mode.

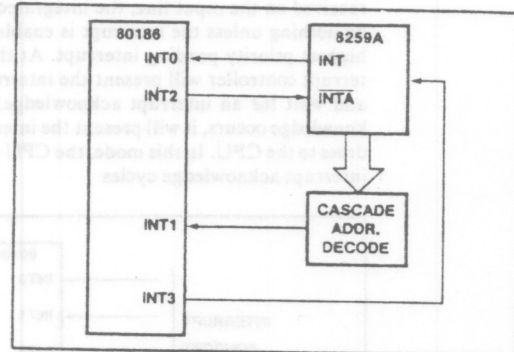


Figure 63. 80186 iRMX86 Mode Interface

Because the integrated interrupt controller is a slave controller, it must be able to generate an interrupt input for an external interrupt controller. It also must be signaled when it has the highest priority pending interrupt to know when to place its interrupt vector on the bus. These two signals are provided by the INT3/Slave Interrupt Output and INT1/Slave Select lines, respectively. The external master interrupt controller must be able to interrupt the 80186 CPU, and needs to know when the interrupt request is acknowledged. The INT0 and INT2/INTA0 lines provide these two functions.

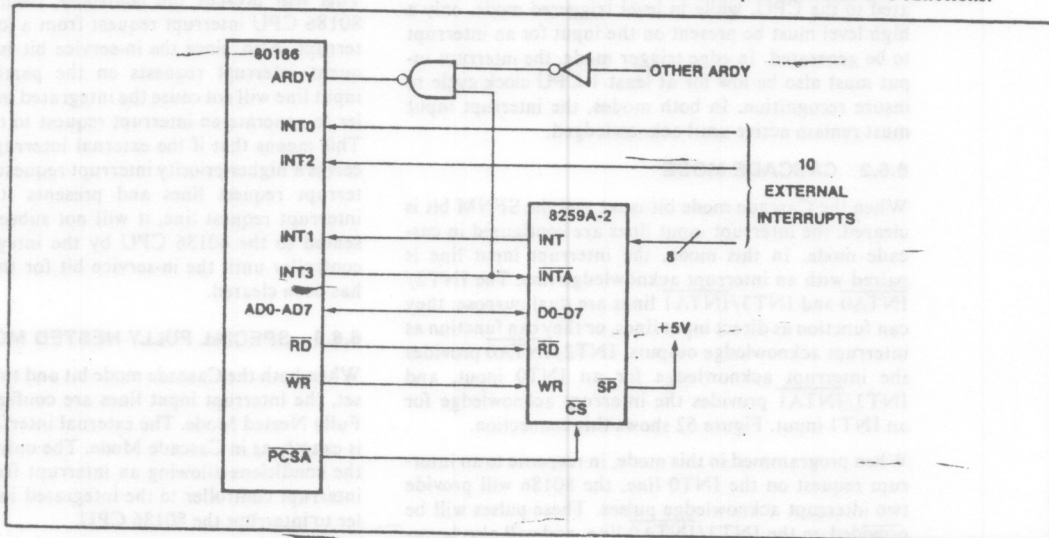


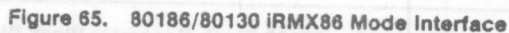
Figure 64. 80186/8259A Interrupt Cascading

Figure 64 shows the 80186 and 8259A in cascade interrupt mode. The code to initialize the 80186 interrupt controller is given in Appendix E. Notice that an "interrupt ready" signal must be returned to the 80186 to prevent the generation of wait states in response to the interrupt acknowledge cycles. In this configuration the INT0 and INT2 lines are used as direct interrupt input lines. Thus, this configuration provides 10 external interrupt lines: 2 provided by the 80186 interrupt controller itself, and 8 from the external 8259A. Also, the 8259A is configured as a master interrupt controller. It will only receive interrupt acknowledge pulses in response to an interrupt it has generated. It may be cascaded again to up to 8 additional 8259As (each of which would be configured in slave mode).

Figure 65 shows the 80186 and 80130 connected in iRMX 86 mode. In this mode, the 80130 interrupt controller is the master interrupt controller of the system.

6.9 Interrupt Latency

Interrupt latency time is the time from when the 80186 receives the interrupt to the time it begins to respond to the interrupt. This is different from interrupt response



time, which is the time from when the processor actually begins processing the interrupt to when it actually executes the first instruction of the interrupt service routine. The factors affecting interrupt latency are the instruction being executed and the state of the interrupt enable flip-flop.

Interrupts will be acknowledged only if the interrupt enable flip-flop in the CPU is set. Thus, interrupt latency will be very long indeed if interrupts are never enabled by the processor!

When interrupts are enabled in the CPU, the interrupt latency is a function of the instructions being executed. Only repeated instructions will be interrupted before being completed, and those only between their respective iterations. This means that the interrupt latency time could be as long as 69 CPU clocks, which is the time it takes the processor to execute an integer divide instruction (with a segment override prefix, see below), the longest single instruction on the 80186.

Other factors can affect interrupt latency. An interrupt will not be accepted between the execution of a prefix (such as segment override prefixes and lock prefixes) and the instruction. In addition, an interrupt will not be accepted between an instruction which modifies any of the segment registers and the instruction immediately following the instruction. This is required to allow the stack to be changed. If the interrupt were accepted, the return address from the interrupt would be placed on a stack which was not valid (the Stack Segment register would have been modified but the Stack Pointer register would not have been). Finally, an interrupt will not be accepted between the execution of the WAIT instruction and the instruction immediately following it if the TEST input is active. If the WAIT sees the TEST input inactive, however, the interrupt will be accepted, and the WAIT will be re-executed after the interrupt return. This is required, since the WAIT is used to prevent execution by the 80186 of an 8087 instruction while the 8087 is busy.

7. CLOCK GENERATOR

The 80186 includes a clock generator which generates the main clock signal for all 80186 integrated components, and all CPU synchronous devices in the 80186 system. This clock generator includes a crystal oscillator, divide by two counter, reset circuitry, and ready generation logic. A block diagram of the clock generator is shown in Figure 66.

7.1 Crystal Oscillator

The 80186 crystal oscillator is a parallel resonant, Pierce oscillator. It was designed to be used as shown in Figure 67. The capacitor values shown are approximate. As the crystal frequency drops, they should be increased, so that at the 4 MHz minimum crystal frequency supported by the 80186 they take on a value of 30pF. The output of this oscillator is not directly available outside the 80186.

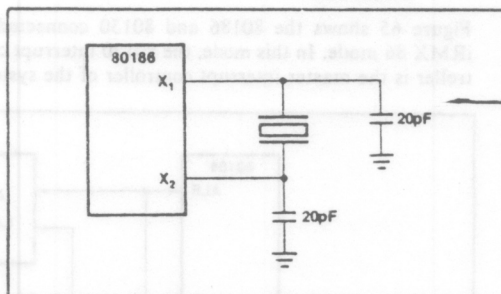


Figure 67. 80186 Crystal Connection

7.2 Using an External Oscillator

An external oscillator may be used with the 80186. The external frequency input (EFI) signal is connected directly to the X1 input of the oscillator. X2 should be left open. This oscillator input is used to drive an internal di-

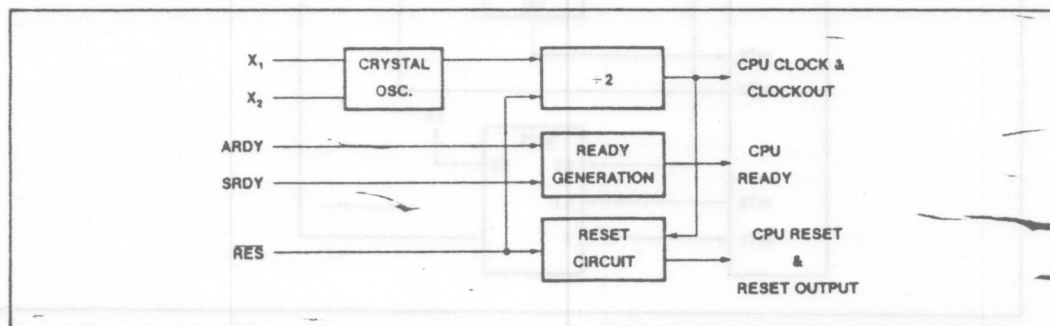


Figure 66. 80186 Clock Generator Block Diagram

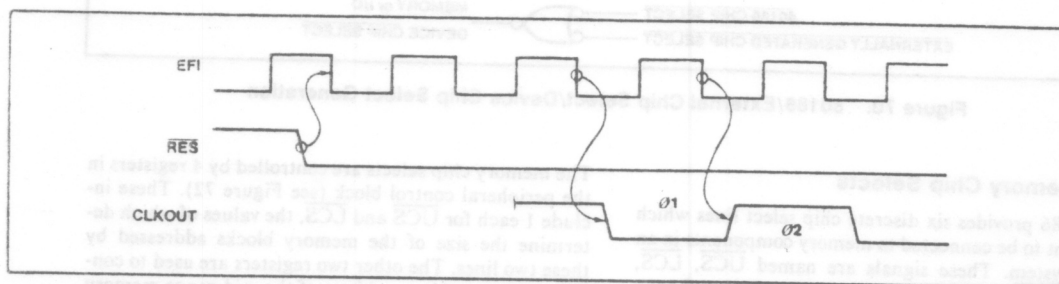


Figure 68. 80186 Clock Generator Reset

vide-by-two counter to generate the CPU clock signal, so the external frequency input can be of practically any duty cycle, so long as the minimum high and low times for the signal (as stated in the data sheet) are met.

7.3 Clock Generator

The output of the crystal oscillator (or the external frequency input) drives a divide by two circuit which generates a 50% duty cycle clock for the 80186 system. All 80186 timing is referenced to this signal, which is available on the CLKOUT pin of the 80186. This signal will change state on the high-to-low transition of the EFI signal.

7.4 Ready Generation

The clock generator also includes the circuitry required for ready generation. Interfacing to the SRDY and ARDY inputs this provides is covered in section 3.1.6.

7.5 Reset

The 80186 clock generator also provides a synchronized reset signal for the system. This signal is generated from the reset input (RES) to the 80186. The clock generator synchronizes this signal to the clockout signal.

The reset input signal also resets the divide-by-two counter. A one clock cycle internal clear pulse is generated when the RES input signal first goes active. This clear pulse goes active beginning on the first low-to-high transition of the X1 input after RES goes active, and goes inactive on the next low-to-high transition of the X1 input. In order to insure that the clear pulse is generated on the next EFI cycle, the RES input signal must satisfy a 25ns setup time to the high-to-low EFI input signal (see Figure 68). During this clear, clockout will be high. On the next high-to-low transition of X1, clockout will go low, and will change state on every subsequent high-to-low transition of EFI.

The reset signal presented to the rest of the 80186, and also the signal present on the RESET output pin of the 80186 is synchronized by the high-to-low transition of the clockout signal of the 80186. This signal remains ac-

tive as long as the $\overline{\text{RES}}$ input also remains active. After the $\overline{\text{RES}}$ input goes inactive, the 80186 will begin to fetch its first instruction (at memory location FFFF0H) after 6 1/2 CPU clock cycles (i.e., T_1 of the first instruction fetch will occur 6 1/2 clock cycles later). To insure that the RESET output will go inactive on the next CPU clock cycle, the inactive going edge of the $\overline{\text{RES}}$ input must satisfy certain hold and setup times to the low-to-high edge of the clockout signal of the 80186 (see Figure 69).

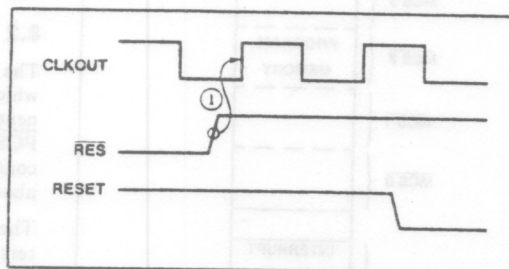


Figure 69. 80186 Coming out of Reset

8. CHIP SELECTS

The 80186 includes a chip select unit which generates hardware chip select signals for memory and I/O accesses generated by the 80186 CPU and DMA units. This unit is programmable such that it can be used to fulfill the chip select requirements (in terms of memory device or bank size and speed) of most small and medium sized 80186 systems.

The chip selects are driven only for internally generated bus cycles. Any cycles generated by an external unit (e.g., an external DMA controller) will not cause the chip selects to go active. Thus, any external bus masters must be responsible for their own chip select generation. Also, during a bus HOLD, the 80186 does not float the chip select lines. Therefore, logic must be included to enable the devices which the external bus master wishes to access (see Figure 70).

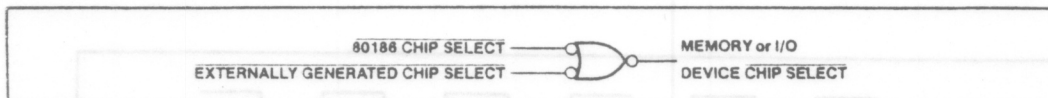


Figure 70. 80186/External Chip Select/Device Chip Select Generation

8.1 Memory Chip Selects

The 80186 provides six discrete chip select lines which are meant to be connected to memory components in an 80186 system. These signals are named $\overline{UCS}$, $\overline{LCS}$, and $\overline{MCS0-3}$ for Upper Memory Chip Select, Lower Memory Chip Select and Midrange Memory Chip Selects 0-3. They are meant (but not limited) to be connected to the three major areas of the 80186 system memory (see Figure 71).

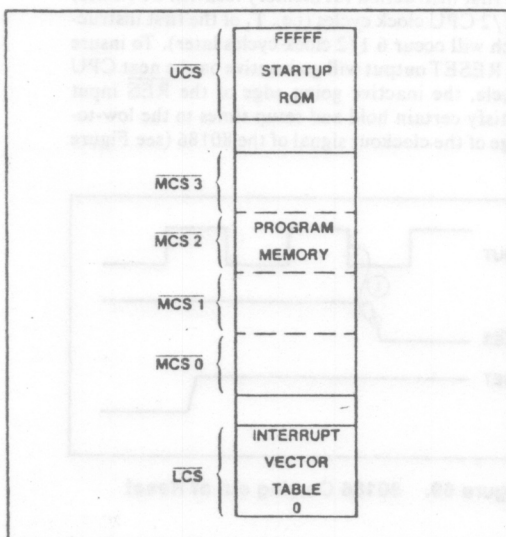


Figure 71. 80186 Memory Areas & Chip Selects

As could be guessed by their names, upper memory, lower memory, and mid-range memory chip selects are designed to address upper, lower, and middle areas of memory in an 80186 system. The upper limit of $\overline{UCS}$ and the lower limit of $\overline{LCS}$ are fixed at FFFFFH and 00000H in memory space, respectively. The other limit of these is set by the memory size programmed into the control register for the chip select line. Mid-range memory allows both the base address and the block size of the memory area to be programmed. The only limitation is that the base address must be programmed to be an integer multiple of the total block size. For example, if the block size was 128K bytes (4 32K byte chunks) the base address could be 0 or 20000H, but not 10000H.

The memory chip selects are controlled by 4 registers in the peripheral control block (see Figure 72). These include 1 each for $\overline{UCS}$ and $\overline{LCS}$, the values of which determine the size of the memory blocks addressed by these two lines. The other two registers are used to control the size and base address of the mid-range memory block.

On reset, only $\overline{UCS}$ is active. It is programmed by reset to be active for the top 1K memory block, to insert 3 wait states to all memory fetches, and to factor external ready for every memory fetch (see section 8.3 for more information on internal ready generation). All other chip select registers assume indeterminate states after reset, but none of the other chip select lines will be active until all necessary registers for a signal have been accessed (not necessarily written, a read to an uninitialized register will enable the chip select function controlled by that register).

8.2 Peripheral Chip Selects

The 80186 provides seven discrete chip select lines which are meant to be connected to peripheral components in an 80186 system. These signals are named $\overline{PCS0-6}$. Each of these lines is active for one of seven contiguous 128 byte areas in memory or I/O space above a programmed base address.

The peripheral chip selects are controlled by two registers in the internal peripheral control block (see Figure 72). These registers allow the base address of the peripherals to be set, and allow the peripherals to be mapped into memory or I/O space. Both of these registers must be accessed before any of the peripheral chip selects will become active.

A bit in the MPCS register allows $\overline{PCS5}$ and $\overline{PCS6}$ to become latched A1 and A2 outputs. When this option is selected, $\overline{PCS5}$ and $\overline{PCS6}$ will reflect the state of A1 and A2 throughout a bus cycle. These are provided to allow external peripheral register selection in a system in which the addresses are not latched. Upon reset, these lines are driven high. They will only reflect A1 and A2 after both PACS and MPCS have been accessed (and are programmed to provide A1 and A2!).

8.3 Ready Generation

The 80186 includes a ready generation unit. This unit generates an internal ready signal for all accesses to memory or I/O areas to which the chip select circuitry of the 80186 responds.

OFFSET:			
A0H	UPPER MEMORY SIZE	①	UMCS
A2H	LOWER MEMORY SIZE	②	LMCS
A4H	PERIPHERAL CHIP SELECT BASE ADDRESS	③	PACS
A6H	MID-RANGE MEMORY BASE ADDRESS	④	MMCS
A8H	MID-RANGE MEMORY SIZE	⑤	MPCS
⑥			
1. Upper memory ready bits 2. Lower memory ready bits 3. PCS0-PCS3 ready bits 4. Mid-range memory ready bits 5. PCS4-PCS6 ready bits 6. MS: 1 = Peripherals active in memory space 0 = Peripherals active in I/O space EX: 1 = 7 PCS lines 0 = PCS5 = A1, PCS6 = A2			
Not all bits of every field are used			

Figure 72. 80186 Chip Select Control Registers

For each ready generation area, 0-3 wait states may be inserted by the internal unit. Table 6 shows how the ready control bits should be programmed to provide this. In addition, the ready generation circuit may be programmed to ignore the state of the external ready (i.e., only the internal ready circuit will be used) or to factor the state of the external ready (i.e., a ready will be returned to the processor only after both the internal ready circuit has gone ready and the external ready has gone ready). Some kind of circuit must be included to generate an external ready, however, since upon reset the ready generator is programmed to factor external ready to all accesses to the top 1K byte memory block. If a ready was not returned on one of the external ready lines (ARDY or SRDY) the processor would wait forever to fetch its first instruction.

Table 6. 80186 Wait State Programming

R2	R1	R0	Number of Wait States
0	0	0	0 + external ready
0	0	1	1 + external ready
0	1	0	2 + external ready
0	1	1	3 + external ready
1	0	0	0 (no external ready required)
1	0	1	1 (no external ready required)
1	1	0	2 (no external ready required)
1	1	1	3 (no external ready required)

8.4 Examples of Chip Select Usage

Many examples of the use of the chip select lines are given in the bus interface section of this note (section 3.2). These examples show how simple it is to use the chip select function provided by the 80186. The key point to remember when using the chip select function is that they are only activated during bus cycles generated by the 80186 CPU or DMA units. When another master has the bus, it must generate its own chip select function. In addition, whenever the bus is given by the 80186 to an external master (through the HOLD/ HLDA arrangement) the 80186 does NOT float the chip select lines.

8.5 Overlapping Chip Select Areas

Generally, the chip selects of the 80186 should not be programmed such that any two areas overlap. In addition, none of the programmed chip select areas should overlap any of the locations of the integrated 256-byte control register block. The consequences of doing this are:

Whenever two chip select lines are programmed to respond to the same area, both will be activated during any access to that area. When this is done, the ready bits for both areas *must* be programmed to the same value. If this is not done, the processor response to an access in this area is indeterminate.

If any of the chip select areas overlap the integrated 256-byte control register block, the timing on the

chip select line is altered. As always, any values returned on the external bus from this access are ignored.

9. SOFTWARE IN AN 80186 SYSTEM

Since the 80186 is object code compatible with the 8086 and 8088, the software in an 80186 system is very similar to that in an 8086 system. Because of the hardware chip select functions, however, a certain amount of initialization code must be included when using those functions on the 80186.

9.1 System Initialization in an 80186 System

Most programmable components of a computer system must be initialized before they are used. This is also true for the 80186. The 80186 includes circuitry which directly affects the ability of the system to address memory and I/O devices, namely the chip select circuitry. This circuitry must be initialized before the memory areas and peripheral devices addressed by the chip select signals are used.

Upon reset, the UMCS register is programmed to be active for all memory fetches within the top 1K byte of memory space. It is also programmed to insert three wait states to all memory accesses within this space. If the hardware chip selects are used, they must be programmed before the processor leaves this 1K byte area of memory. If a jump to an area for which the chips are not selected occurs, the microcomputer system will cease to operate (since the processor will fetch garbage from the data bus). Appendix F shows a typical initialization sequence for the 80186 chip select unit.

Once the chip selects have been properly initialized, the rest of the 80186 system may be initialized much like an 8086 system. For example, the interrupt vector table might get set up, the interrupt controller initialized, a serial I/O channel initialized, and the main program begun. Note that the integrated peripherals included in the 80186 do not share the same programming model as the standard Intel peripherals used to implement these functions in a typical 8086 system, i.e., different values must be programmed into different registers to achieve the same function using the integrated peripherals. Appendix F shows a typical initialization sequence for an interrupt driven system using the 80186 interrupt controller.

9.2 Initialization for iRMX™ 86 System

Using the iRMX 86 operating system with the 80186 requires an external 8259A and an external 8253/4 or alternatively an external 80130 OSF component. These are required because the operating system is interrupt driven, and expects the interrupt controller and timers to have the register model of these external devices. This

model is not the same as is implemented by the 80186. Because of this, the 80186 interrupt controller must be placed in iRMX 86 mode after reset. This initialization can be done at any time after reset before jump to the root task of iRMX 86 System is actually performed. If need be, a small section of code which initializes both the 80186 chip selects and the 80186 interrupt controller can be inserted between the reset vector location and the beginning of iRMX 86 System (see Figure 73). In this case, upon reset, the processor would jump to the 80186 initialization code, and when this has been completed, would jump to the iRMX 86 initialization code (in the root task). It is important that the 80186 hardware be initialized before iRMX 86 operation is begun, since some of the resources addressed by the 80186 system may not be initialized properly by iRMX 86 System if the initialization is done in the reverse manner.

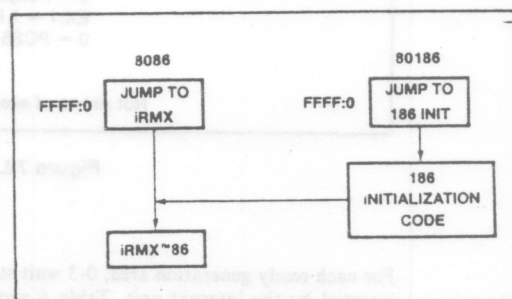


Figure 73. iRMX-86 Initialization with 8086 & 80186

9.3 Instruction Execution Differences Between the 8086 and 80186

There are a few instruction execution differences between the 8086 and the 80186. These differences are:

Undefined Opcodes:

When the opcodes 63H, 64H, 65H, 66H, 67H, F1H, FEH, XX1111XXB and FFH, XX1111XXB are executed, the 80186 will execute an illegal instruction exception, interrupt type 6. The 8086 will ignore the opcode.

0FH opcode:

When the opcode 0FH is encountered, the 8086 will execute a POP CS, while the 80186 will execute an illegal instruction exception, interrupt type 6.

Word Write at Offset FFFFH:

When a word write is performed at offset FFFFH in a segment, the 8086 will write one byte at offset FFFFH, and the other at offset 0, while the 80186 will write one byte at offset

FFFFH, and the other at offset 10000H (one byte beyond the end of the segment). One byte segment underflow will also occur (on the 80186) if a stack PUSH is executed and the Stack Pointer contains the value 1.

Shift/Rotate by Value Greater Than 31:

Before the 80186 performs a shift or rotate by a value (either in the CL register, or by an immediate value) it ANDs the value with 1FH, limiting the number of bits rotated to less than 32. The 8086 does not do this.

LOCK prefix:

The 8086 activates its LOCK signal immediately after executing the LOCK prefix. The 80186 does not activate the LOCK signal until the processor is ready to begin the data cycles associated with the LOCKed instruction.

Interrupted String Move Instructions:

If an 8086 is interrupted during the execution of a repeated string move instruction, the return value it will push on the stack will point to the last prefix instruction before the string move instruction. If the instruction had more than one prefix (e.g., a segment override prefix in addition to the repeat prefix), it will not be re-executed upon returning from the interrupt. The 80186 will push the value of the first prefix to the repeated instruction, so long as prefixes are not repeated, allowing the string instruction to properly resume.

Conditions causing divide error with an integer divide:

The 8086 will cause a divide error whenever the absolute value of the quotient is greater than 7FFFH (for word operations) or if the absolute value of the quotient is greater than 7FH (for byte operations). The 80186 has expanded the range of negative numbers allowed as a quotient

by 1 to include 8000H and 80H. These numbers represent the most negative numbers representable using 2's complement arithmetic (equaling -32768 and -128 in decimal, respectively).

ESC Opcode:

The 80186 may be programmed to cause an interrupt type 7 whenever an ESCape instruction (used for co-processors like the 8087) is executed. The 8086 has no such provision. Before the 80186 performs this trap, it must be programmed to do so.

These differences can be used to determine whether the program is being executed on an 8086 or an 80186. Probably the safest execution difference to use for this purpose is the difference in multiple bit shifts. For example, if a multiple bit shift is programmed where the shift count (stored in the CL register!) is 33, the 8086 will shift the value 33 bits, whereas the 80186 will shift it only a single bit.

In addition to the instruction execution differences noted above, the 80186 includes a number of new instruction types, which simplify assembly language programming of the processor, and enhance the performance of higher level languages running on the processor. These new instructions are covered in depth in the 8086/80186 users manual and in appendix H of this note.

10. CONCLUSIONS

The 80186 is a glittering example of state-of-the art integrated circuit technology applied to make the job of the microprocessor system designer simpler and faster. Because many of the required peripherals and their interfaces have been cast in silicon, and because of the timing and drive latitudes provided by the part, the designer is free to concentrate on other issues of system design. As a result, systems designed around the 80186 allow applications where no other processor has been able to provide the necessary performance at a comparable size or cost.

3-312

APPENDIX A: PERIPHERAL CONTROL BLOCK

All the integrated peripherals within the 80186 microprocessor are controlled by sets of registers contained within an integrated peripheral control block. The registers are physically located within the peripheral devices they control, but are addressed as a single block of registers. This set of registers fills 256 contiguous bytes and can be located beginning on any 256 byte boundary of the 80186 memory or I/O space. A map of these registers is shown in Figure A-1.

A.1 Setting the Base Location of the Peripheral Control Block

In addition to the control registers for each of the integrated 80186 peripheral devices, the peripheral control

block contains the peripheral control block relocation register. This register allows the peripheral control block to be re-located on any 256 byte boundary within the processor's memory or I/O space. Figure A-2 shows the layout of this register.

This register is located at offset FEH within the peripheral control block. Since it is itself contained within the peripheral control block, any time the location of the peripheral control block is moved, the location of the relocation register will also move.

In addition to the peripheral control block relocation information, the relocation register contains two additional bits. One is used to set the interrupt controller into iRMX86 compatibility mode. The other is used to force the processor to trap whenever an ESCape (coprocessor) instruction is encountered.

	OFFSET
Relocation Register	FEH
DMA Descriptors Channel 1	DAH
	D0H
DMA Descriptors Channel 0	CAH
	C0H
Chip-Select Control Registers	A8H
	A0H
Timer 2 Control Registers	66H
	60H
Timer 1 Control Registers	5EH
	58H
Timer 0 Control Registers	56H
	50H
Interrupt Controller Registers	3EH
	20H

Figure A-1. 80186 Integrated Peripheral Control Block

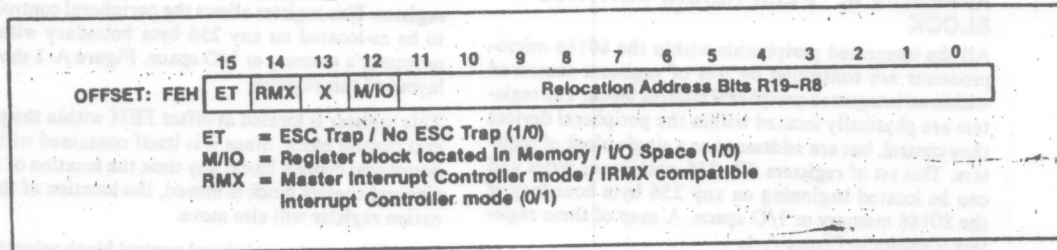


Figure A-2. 80186 Relocation Register Layout

Because the relocation register is contained within the peripheral control block, upon reset the relocation register is automatically programmed with the value 20FFH. This means that the peripheral control block will be located at the very top (FF00H to FFFFH) of I/O space. Thus, after reset the relocation register will be located at word location FFFEh in I/O space.

If the user wished to locate the peripheral control block starting at memory location 10000H he would program the peripheral control register with the value 1100H. By doing this, he would move all registers within the integrated peripheral control block to memory locations 10000H to 100FFH. Note that since the relocation register is contained within the peripheral control block, it too would move to word location 100FEH in memory space.

A.2 Peripheral Control Block Registers

Each of the integrated peripherals' control and status registers are located at a fixed location above the programmed base location of the peripheral control block. There are many locations within the peripheral control block which are not assigned to any peripheral. If a write is made to any of these locations, the bus cycle will be run, but the value will not be stored in any internal location. This means that if a subsequent read is made to the same location, the value written will not be read back.

The processor will run an external bus cycle for any memory or I/O cycle which accesses a location within the integrated control block. This means that the address, data, and control information will be driven on the 80186 external pins just as if a "normal" bus cycle had been run. Any information returned by an external device will be ignored, however, even if the access was to a location which does not correspond to any of the inte-

grated peripheral control registers. The above is also true for the 80188, except that the word access made to the integrated registers will be performed in a single bus cycle, with only the lower 8 bits of data being driven by the write cycle (since the upper 8 bits of data are non-existent on the external data bus!)

The processor internally generates a ready signal whenever any of the integrated peripherals are accessed; thus any external ready signals are ignored whenever an access is made to any location within the integrated peripheral register control block. This ready will also be returned if an access is made to a location within the 256 byte area of the peripheral control block which does not correspond to any integrated peripheral control register. The processor will insert 0 wait states to any access within the integrated peripheral control block except for accesses to the timer registers. ANY access to the timer control and counting registers will incur 1 wait state. This wait state is required to properly multiplex processor and counter element accesses to the timer control registers.

All accesses made to the integrated peripheral control block must be WORD accesses. Any write to the integrated registers will modify all 16 bits of the register, whether the opcode specified a byte write or a word write. A byte read from an even location should cause no problems, but the data returned when a byte read is performed from an odd address within the peripheral control block is undefined. This is true both for the 80186 AND the 80188. As stated above, even though the 80188 has an external 8 bit data bus, internally it is still a 16 bit machine. Thus, the word accesses performed to the integrated registers by the 80188 will each occur in a single bus cycle with only the lower 8 bits of data being driven on the external data bus (on a write).

APPENDIX B: 80186 SYNCHRONIZATION INFORMATION

Many input signals to the 80186 are asynchronous, that is, a specified set up or hold time is not required to insure proper functioning of the device. Associated with each of these inputs is a synchronizer which samples this external asynchronous signal, and synchronizes it to the internal 80186 clock.

B.1 Why Synchronizers Are Required

Every data latch requires a certain set up and hold time in order to operate properly. At a certain window within the specified set up and hold time, the part will actually try to latch the data. If the input makes a transition within this window, the output will not attain a stable state within the given output delay time. The size of this sampling window is typically much smaller than the actual window specified by the data sheet, however part to part variation could move this window around within the specified window in the data sheet.

Even if the input to a data latch makes a transition while a data latch is attempting to latch this input, the output of the latch will attain a stable state after a certain amount of time, typically much longer than the normal strobe to output delay time. Figure B-1 shows a normal input to output strobed transition and one in which the input signal makes a transition during the latch's sample window. In order to synchronize an asynchronous signal, all one needs to do is to sample the signal into one data latch, wait a certain amount of time, then latch it into a second data latch. Since the time between the strobe into the first data latch and the strobe into the second data latch allows the first data latch to attain a steady state (or to resolve the asynchronous signal), the second data latch will be presented with an input signal which satisfies any set up and hold time requirements it may have. Thus, the output of this second latch is a synchronous signal with respect to its strobe input.

A synchronization failure can occur if the synchronizer fails to resolve the asynchronous transition within the time between the two latch's strobe signals. The rate of failure is determined by the actual size of the sampling

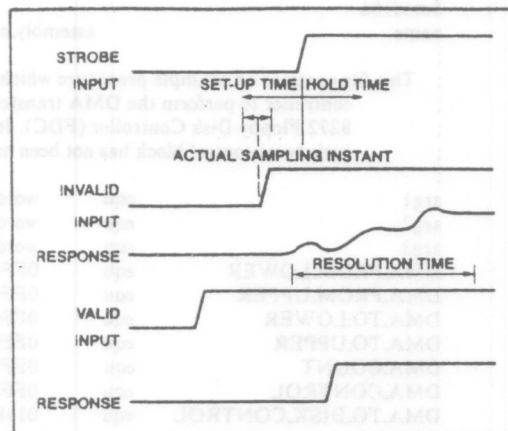


Figure B-1. Valid & Invalid Latch Input Transitions & Responses

window of the data latch, and by the amount of time between the strobe signals of the two latches. Obviously, as the sampling window gets smaller, the number of times an asynchronous transition will occur during the sampling window will drop. In addition, however, a smaller sampling window is also indicative of a faster resolution time for an input transition which manages to fall within the sampling window.

B.2 80186 Synchronizers

The 80186 contains synchronizers on the $\overline{\text{RES}}$, $\overline{\text{TEST}}$, TmrIn0-1 , DRQ0-1 , NMI , INT0-3 , ARDY , and HOLD input lines. Each of these synchronizers use the two stage synchronization technique described above (with some minor modifications for the ARDY line, see section 3.1.6). The sampling window of the latches is designed to be in the tens of pico-seconds, and should allow operation of the synchronizers with a mean time between failures of over 30 years assuming continuous operation.

APPENDIX C: 80186 EXAMPLE DMA INTERFACE CODE

```

$mod186
name                assembly.example.80186.DMA_support
;
; This file contains an example procedure which initializes the 80186 DMA
; controller to perform the DMA transfers between the 80186 system the the
; 8272 Floppy Disk Controller (FDC). It assumes that the 80186
; peripheral control block has not been moved from its reset location.
;
arg1                equ     word ptr [BP + 4]
arg2                equ     word ptr [BP + 6]
arg3                equ     word ptr [BP + 8]
DMA.FROM.LOWER      equ     0FFC0h                ; DMA register locations
DMA.FROM.UPPER      equ     0FFC2h
DMA.TO.LOWER        equ     0FFC4h
DMA.TO.UPPER        equ     0FFC6h
DMA.COUNT           equ     0FFC8h
DMA.CONTROL         equ     0FFCAh
DMA.TO.DISK_CONTROL equ     01486h                ; destination synchronization
; source to memory, incremented
; destination to I/O
; no terminal count
; byte transfers
DMA.FROM.DISK_CONTROL equ 0A046h                ; source synchronization
; source to I/O
; destination to memory, incr
; no terminal count
; byte transfers
FDC.DMA             equ     6B8h                ; FDC DMA address
FDC.DATA            equ     688h                ; FDC data register
FDC.STATUS          equ     680h                ; FDC status register
;
cgroup              group   code
code                segment public set_dma_
; assume cs:cgroup
;
; set_dma (offset,to) programs the DMA channel to point one side to the
; disk DMA address, and the other to memory pointed to by ds:offset. If
; 'to' = 0 then will be a transfer from disk to memory; if
; 'to' = 1 then will be a transfer from memory to disk. The parameters to
; the routine are passed on the stack.
;
set_dma             proc     near
enter              0,0                ; set stack addressability
push               AX                  ; save registers used
push               BX
push               DX
test               arg2,1              ; check to see direction of
; transfer
jz                 from_disk
; performing a transfer from memory to the disk controller
;
mov                AX,DS               ; get the segment value
rol                AX,4                ; gen the upper 4 bits of the
; physical address in the lower 4
; bits of the register

```



```

mov     BX,AX                ; save the result...
mov     DX,DMA.FROM.UPPER    ; prgm the upper 4 bits of the
out     DX,AX                ; DMA source register
and     AX,0FFF0h            ; form the lower 16 bits of the
                                ; physical address
add     AX,arg1              ; add the offset
mov     DX,DMA.FROM.LOWER    ; prgm the lower 16 bits of the
out     DX,AX                ; DMA source register
jnc     no.carry.from        ; check for carry out of addition
inc     BX                   ; if carry out, then need to adj
mov     AX,BX                ; the upper 4 bits of the pointer
mov     DX,DMA.FROM.UPPER
out     DX,AX

no.carry.from:
mov     AX,FDC.DMA           ; prgm the low 16 bits of the DMA
mov     DX,DMA.TO.LOWER      ; destination register
out     DX,AX
xor     AX,AX                ; zero the up 4 bits of the DMA
mov     DX,DMA.TO.UPPER      ; destination register
out     DX,AX
mov     AX,DMA.TO.DISK.CONTROL; prgm the DMA ctl reg
mov     DX,DMA.CONTROL       ; note: DMA may begin immediatly
out     DX,AX                ; after this word is output
pop     DX
pop     BX
pop     AX
leave
ret

from.disk:
;
; performing a transfer from the disk to memory
;

mov     AX,DS
rol     AX,4
mov     DX,DMA.TO.UPPER
out     DX,AX
mov     BX,AX
and     AX,0FFF0h
add     AX,arg1
mov     DX,DMA.TO.LOWER
out     DX,AX
jnc     no.carry.to
inc     BX
mov     AX,BX
mov     DX,DMA.TO.UPPER
out     DX,AX

no.carry.to:
mov     AX,FDC.DMA
mov     DX,DMA.FROM.LOWER
out     DX,AX
xor     AX,AX
mov     DX,DMA.FROM.UPPER
out     DX,AX
mov     AX,DMA.FROM.DISK.CONTROL
mov     DX,DMA.CONTROL

```


APPENDIX D: 80186 EXAMPLE TIMER INTERFACE CODE

```

$mod186
name
    example.80186.timer.code
;
; this file contains example 80186 timer routines. The first routine
; sets up the timer and interrupt controller to cause the timer
; to generate an interrupt every 10 milliseconds, and to service
; interrupt to implement a real time clock. Timer 2 is used in
; this example because no input or output signals are required.
; The code example assumes that the peripheral control block has
; not been moved from its reset location (FF00-FFFF in I/O space).
;
arg1 equ word ptr [BP + 4]
arg2 equ word ptr [BP + 6]
arg3 equ word ptr [BP + 8]
timer2int equ 19 ; timer 2 has vector type 19
timer2control equ 0FF66h
timer2max_ctl equ 0FF62h
timer_int_ctl equ 0FF32h ; interrupt controller regs
eoi_register equ 0FF22h
interrupt_stat equ 0FF30h

data segment public 'data'
    public hour_, minute_, second_, msec_
    db ?
    db ?
    db ?
    db ?
data ends

cgroup group code
dgroup group data

code segment public 'code'
    public set_time
    assume cs:code, ds:dgroup
;
; set_time(hour, minute, second) sets the time variables, initializes the
; 80186 timer2 to provide interrupts every 10 milliseconds, and
; programs the interrupt vector for timer 2
;
set_time proc near
    enter 0,0 ; set stack addressability
    push AX ; save registers used
    push DX
    push SI
    push DS

    xor AX, AX ; set the interrupt vector
    ; the timers have unique
    ; interrupt
    ; vectors even though they share
    ; the same control register

    mov DS, AX

    mov SI, 4 * timer2int

```



```

mov     DS:[SI],offset timer2_interrupt_routine
inc     SI
inc     SI
mov     DS:[SI],CS
pop     DS

mov     AX,arg1           ; set the time values
mov     hour,,AL
mov     AX,arg2
mov     minute,,AL
mov     AX,arg3
mov     second,,AL
mov     msec,,0

mov     DX,timer2_maxctl  ; set the max count value
mov     AX,20000          ; 10 ms / 500 ns (timer 2 counts
                        ; at 1/4 the CPU clock rate)

out     DX,AX
mov     DX,timer2_control ; set the control word
mov     AX,111000000000001b ; enable counting
                        ; generate interrupts on TC
                        ; continuous counting

out     DX,AX

mov     DX,timer_intctl   ; set up the interrupt controller
mov     AX,0000b         ; unmask interrupts
                        ; highest priority interrupt

out     DX,AX
sti     ; enable processor interrupts

pop     SI
pop     DX
pop     AX
leave
ret
endp

set_time:

timer2_interrupt_routine proc far
push    AX
push    DX

cmp     msec,,99          ; see if one second has passed
jae     bump_second      ; if above or equal...

bump_second:
mov     msec,,0           ; reset millisecond
cmp     second,,59        ; see if one minute has passed
jae     bump_minute

bump_minute:
mov     second,,0
cmp     minute,,59        ; see if one hour has passed
jae     bump_hour
inc     minute
jmp     reset_intctl

mov     second,,0
cmp     minute,,59
jae     bump_hour
inc     minute
jmp     reset_intctl

mov     second,,0
cmp     minute,,59
jae     bump_hour
inc     minute
jmp     reset_intctl

mov     second,,0
cmp     minute,,59
jae     bump_hour
inc     minute
jmp     reset_intctl

```

bump.hour:

```

mov     minute,,0
cmp     hour,,12           ; see if 12 hours have passed
jae     reset.hour
inc     hour.
jmp     resetIntCtl

```

reset.hour:

```

mov     hour,,1

```

resetIntCtl:

```

mov     DX,coi.register
mov     AX,8000h           ; non-specific end of interrupt
out     DX,AX

pop     DX
pop     AX
iret

timer2.interrupt_routine
code
ends
end

```

\$mod186

name

example.80186.baud.code

;

```

; this file contains example 80186 timer routines. The second routine
; sets up the timer as a baud rate generator. In this mode,
; Timer 1 is used to continually output pulses with a period of
; 6.5 usec for use with a serial controller at 9600 baud
; programmed in divide by 16 mode (the actual period required
; for 9600 baud is 6.51 usec). This assumes that the 80186 is
; running at 8 MHz. The code example also assumes that the
; peripheral control block has not been moved from its reset
; location (FF00-FFFF in I/O space).

```

```

timer1.control      equ     0FF5Eh
timer1.max_cnt      equ     0FF5Ah

```

code

```

segment
assume     cs:code

```

public 'code'

;

```

; set_baud() initializes the 80186 timer 1 as a baud rate generator for
; a serial port running at 9600 baud

```

;

set_baud.

```

proc     near
push     AX                ; save registers used
push     DX

mov     DX,timer1.max_cnt  ; set the max count value
mov     AX,13              ; 500ns * 13 = 6.5 usec
out     DX,AX

mov     DX,timer1.control  ; set the control word
mov     AX,110000000000001b ; enable counting
; no interrupt on TC
; continuous counting
; single max count register

out     DX,AX

pop     DX
pop     AX

```

```

ret
set_baud      endp
code          ends
end

$mod186
name          example.80186.count.code
;
; this file contains example 80186 timer routines. The third routine
; sets up the timer as an external event counter. In this mode,
; Timer 1 is used to count transitions on its input pin. After
; the timer has been set up by the routine, the number of
; events counted can be directly read from the timer count
; register at location FF58H in I/O space. The timer will
; count a maximum of 65535 timer events before wrapping
; around to zero. This code example also assumes that the
; peripheral control block has not been moved from its reset
; location (FF00-FFFF in I/O space).
;
timer1.control equ 0FF5EH
timer1.max_cnt equ 0FF5AH
timer1.cnt_reg equ 0FF58H

code          segment                public 'code'
assume        cs:code
;
; set_count() initializes the 80186 timer1 as an event counter
;
set_count     proc near
push          AX                    ; save registers used
push          DX
mov           DX,timer1.max_cnt    ; set the max count value
mov           AX,0                  ; allows the timer to count
                                           ; all the way to FFFFH
out           DX,AX
mov           DX,timer1.control    ; set the control word
mov           AX,1100000000000101b ; enable counting
                                           ; no interrupt on TC
                                           ; continuous counting
                                           ; single max count register
                                           ; external clocking
out           DX,AX
xor           AX,AX                ; zero AX
mov           DX,timer1.cnt_reg    ; and zero the count in the timer
out           DX,AX                ; count register
pop           DX
pop           AX
ret
set_count     endp
code          ends
end

```


APPENDIX E: 80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

```

$mod186
name
    example.80186.interrupt_code
;
; This routine configures the 80186 interrupt controller to provide
; two cascaded interrupt inputs (through an external 8259A
; interrupt controller on pins INT0/INT2) and two direct
; interrupt inputs (on pins INT1 and INT3). The default priority
; levels are used. Because of this, the priority level programmed
; into the control register is set the 111, the level all
; interrupts are programmed to at reset.
;
int0_control    equ    0FF38H
int_mask        equ    0FF28H
;
code            segment
    assume      CS:code
    proc        near
    push        DX
    push        AX

    mov         AX,0100111B
; cascade mode
; interrupt unmasked

    mov         DX,int0_control
    out         DX,AX

    mov         AX,01001101B
; now unmask the other external
; interrupts

    mov         DX,int_mask
    out         DX,AX
    pop         AX
    pop         DX
    ret
endp
set_int_code    code
end

$mod186
name
    example.80186.interrupt_code
;
; This routine configures the 80186 interrupt controller into iRMX 86
; mode. This code does not initialize any of the 80186
; integrated peripheral control registers, nor does it initialize
; the external 8259A or 80130 interrupt controller.
;
relocation_reg  equ    0FFFEH
;
code            segment
    assume      CS:code
    proc        near
    push        DX
    push        AX

    mov         DX,relocation_reg
    in          AX,DX
    or          AX,0100000000000000B
; read old contents of register
; set the RMX mode bit
    out         DX,AX

```

set_rmx.
code

pop AX
pop DX
ret
endp
ends
end

APPENDIX E: 80186 EXAMPLE INTERRUPT CONTROLLER INTERFACE CODE

Example 80186 interrupt code

This routine configures the 80186 interrupt controller to provide two cascaded interrupt inputs (through an external 8259A interrupt controller on pins INT0/INT2) and two direct interrupt inputs (on pins INT1 and INT3). The default priority levels are used. Because of this, the priority level programmed into the control register is set the 11, the level all interrupts are programmed to occur.

int_control
int_mask

org 0FF18H
org 0FF19H

public code

segment
assume CS:code
proc
push DX
push AX
ret

cascade mode
interrupts requested

mov AX,010011B
mov DX,int_control
out DX,AX

now cascade the other external
interrupts

mov AX,0100101B
mov DX,int_mask
out DX,AX

pop DX
pop AX
ret
crlf
end
end

set_irq
code

Example 80186 interrupt code

This routine configures the 80186 interrupt controller into IRMX 86 mode. This code does not initialize any of the 80186 integrated peripheral control registers, nor does it initialize the external 8259A or 80180 interrupt controller.

org 0FF18H

public code

segment
assume CS:code
proc
push DX
push AX
ret

read old contents of register
set the IRMX mode bit

mov BX,00000000H
in AX,BX
or AX,01000000H
out BX,AX

APPENDIX F: 80186/8086 EXAMPLE SYSTEM INITIALIZATION CODE

```

name                example.80186.system.init
;
; This file contains a system initialization routine for the 80186
; or the 8086. The code determines whether it is running on
; an 80186 or an 8086, and if it is running on an 80186, it
; initializes the integrated chip select registers.
;
restart              segment at                                0FFFFh
; This is the processor reset address at 0FFFF0H
;
; org 0
; jmp far ptr initialize
restart              ends
;
; extrn monitor:far
initLhw              segment at                                0FFFF0h
; assume CS:initLhw
;
; This segment initializes the chip selects. It must be located in the
; top 1K to insure that the ROM remains selected in the 80186
; system until the proper size of the select area can be programmed.
;
UMCS.reg              equ 0FFA0H                               ; chip select register locations
LMCS.reg              equ 0FFA2H
PACS.reg              equ 0FFA4H
MPCS.reg              equ 0FFA8H
UMCS.value            equ 0F800H                               ; 64K, no wait states
LMCS.value            equ 07F8H                               ; 32K, no wait states
PACS.value            equ 72H                                  ; peripheral base at 400H, 2 ws
MPCS.value            equ 0BAH                                  ; PCS5 and 6 supplies,
; peripherals in I/O space
;
initialize            proc far
; mov AX,2                                                     ; determine if this is an
; mov CL,33                                                     ; 8086 or an 80186 (checks
; shr AX,CL                                                     ; to see if the multiple bit
; test AX,1                                                     ; shift value was ANDed)
; jz notL80186
;
; mov DX,UMCS.reg                                               ; program the UMCS register
; mov AX,UMCS.value
; out DX,AX
;
; mov DX,LMCS.reg                                               ; program the LMCS register
; mov AX,LMCS.value
; out DX,AX
;
; mov DX,PACS.reg                                               ; set up the peripheral chip
;                                                         ; selects (note the mid-range
;                                                         ; memory chip selects are not
;                                                         ; needed in this system, and
;                                                         ; are thus not initialized
;
; mov AX,PACS.value
; out DX,AX

```



```

mov    DX,MPCS.reg
mov    AX,MPCS.value
out    DX,AX

```

Now that the chip selects are all set up, the main program of the computer may be executed.

not.80186:

```

initialize
init_hw

```

```

jmp    far ptr monitor
endp
ends
end

```

APPENDIX G: 80186 WAIT STATE PERFORMANCE

Because the 80186 contains separate bus interface and execution units, the actual performance of the processor will not degrade at a constant rate as wait states are added to the memory cycle time from the processor. The actual rate of performance degradation will depend on the type and mix of instructions actually encountered in the user's program.

Shown below are two 80186 assembly language programs, and the actual execution time for the two programs as wait states are added to the memory system of the processor. These programs show the two extremes to which wait states will or will not effect system performance as wait states are introduced.

Program 1 is very memory intensive. It performs many memory reads and writes using the more extensive memory addressing modes of the processor (which also take a greater number of bytes in the opcode for the instruction). As a result, the execution unit must constantly wait for the bus interface unit to fetch and perform the memory cycles to allow it to continue. Thus, the execution time of this type of routine will grow quickly as wait states are added, since the execution time is almost totally limited to the speed at which the processor can run bus cycles.

Note also that this program execution times calculated by merely summing up the number of clock cycles given in the data sheet will typically be less than the actual number of clock cycles actually required to run the program. This is because the numbers quoted in the data sheet assume that the opcode bytes have been prefetched and reside in the 80186 prefetch queue for immediate access by the execution unit. If the execution unit cannot

access the opcode bytes immediately upon request, dead clock cycles will be inserted in which the execution unit will remain idle, thus increasing the number of clock cycles required to complete execution of the program.

On the other hand, program 2 is more CPU intensive. It performs many integer multiplies, during which time the bus interface unit can fill up the instruction prefetch queue in parallel with the execution unit performing the multiply. In this program, the bus interface unit can perform bus operations faster than the execution unit actually requires them to be run. In this case, the performance degradation is much less as wait states are added to the memory interface. The execution time of this program is closer to the number of clock cycles calculated by adding the number of cycles per instruction because the execution unit does not have to wait for the bus interface unit to place an opcode byte in the prefetch queue as often. Thus, fewer clock cycles are wasted by the execution unit laying idle for want of instructions. Table G-1 lists the execution times measured for these two programs as wait states were introduced with the 80186 running at 8 MHz.

Table G-1

# of Wait States	Program 1		Program 2	
	Exec Time (μsec)	Perf Degr	Exec Time (μsec)	Perf Degr
0	505		294	
1	595	18%	311	6%
2	669	12%	337	8%
3	752	12%	347	3%

\$mod186

name

example_wait_state_performance

```
; This file contains two programs which demonstrate the 80186 performance
; degradation as wait states are inserted. Program 1 performs a
; transformation between two types of characters sets, then copies
; the transformed characters back to the original buffer (which is 64
; bytes long. Program 2 performs the same type of transformation, however
; instead of performing a table lookup, it multiplies each number in the
; original 32 word buffer by a constant (3, note the use of the integer
; immediate multiply instruction). Program "nothing" is used to measure
; the call and return times from the driver program only.
```

cgroup
dgroup
data

group code
group data
segment

public 'data'

lstring	db
marray	dw
data	ends

```
code          segment          public 'code'
              assume CS:cgroup,DS:dgroup
              public bench_1,bench_2,nothing_wait_state_set_timer.
```

```

bench.1      proc      near
              push     SI          ; save registers used
              push     CX
              push     BX
              push     AX

```

```
mov     CX,64          ; translate 64 bytes
mov     SI,0
mov     BH,0
```

```

loop.back:      mov     BL,Lstring[SI]      ; get the byte
                mov     AL,Ltable[BX]      ; translate byte
                mov     Lstring[SI],AL     ; and store it
                inc     SI                 ; increment index
                loop     loop.back         ; do the next byte

```

Program 1		Program 2	
Exec (Time)	Part (Page)	Exec (Time)	Part (Page)
105		104	
205		bench_1	85
409		bench_2	85
513		103	10

Exec Time (sec)	Part	Exec Time (sec)	Part	pop	AX
305		305		pop	BX
305		305		pop	CX
305		305		pop	SI
305	bench.1	305		ret	
305		305		endp	
609	bench.2	609		proc	near
135		135		push	AX
				push	SI
				push	CX

```
mov     CX,32                                ; multiply 32 numbers
mov     SI,offset m_array
```

```

loop_back2:
    imul    AX,word ptr [SI],3      ; immediate multiply
    mov     word ptr [SI],AX
    inc     SI
    inc     SI
    loop    loop_back2

```

bench_2	pop	CX
	pop	SI
	pop	AX
	ret	
	endp	


```

nothing.      proc      near
               ret
nothing.      endp
;
; wait_state(n) sets the 80186 LMCS register to the number of wait states
; (0 to 3) indicated by the parameter n (which is passed on the stack).
; No other bits of the LMCS register are modified.
;
wait_state.   proc      near
               enter     0,0           ; set up stack frame
               push      AX           ; save registers used
               push      BX
               push      DX

               mov       BX,word ptr [BP + 4] ; get argument
               mov       DX,0FFA2h        ; get current LMCS register

contents      in        AX,DX

               and       AX,0FFFCh        ; and off existing ready bits
               and       BX,3             ; insure ws count is good
               or        AX,BX            ; adjust the ready bits
               out       DX,AX            ; and write to LMCS

               pop       DX
               pop       BX
               pop       AX
               leave      ; tear down stack frame
               ret
wait_state.   endp
;
; set_timer() initializes the 80186 timers to count microseconds. Timer 2
; is set up as a prescaler to timer 0, the microsecond count can be read
; directly out of the timer 0 count register at location FF50H in I/O
; space.
;
set_timer.    proc      near
               push      AX
               push      DX

               mov       DX,0ff66h        ; stop timer 2
               mov       AX,4000h
               out       DX,AX

               mov       DX,0ff50h        ; clear timer 0 count
               mov       AX,0
               out       DX,AX

               mov       DX,0ff52h        ; timer 0 counts up to 65535
               mov       AX,0
               out       DX,AX

```

```
mov     DX,0ff56h           ; enable timer 0
mov     AX,0c009h
out     DX,AX

mov     DX,0ff60h           ; clear timer 2 count
mov     AX,0
out     DX,AX

mov     DX,0ff62h           ; set maximum count of timer 2
mov     AX,2
out     DX,AX

mov     DX,0ff66h           ; re-enable timer 2
mov     AX,0c001h
out     DX,AX

pop     DX
pop     AX
ret
endp
ends
end
```

set_timer.
code

APPENDIX H: 80186 NEW INSTRUCTIONS

The 80186 performs many additional instructions to those of the 8086. These instructions appear shaded in the instruction set summary at the back of the 80186 data sheet. This appendix explains the operation of these new instructions. In order to use these new instructions with the 8086/186 assembler, the "\$mod186" switch must be given to the assembler. This can be done by placing the line: "\$mod186" at the beginning of the assembly language file.

• PUSH immediate

This instruction allows immediate data to be pushed onto the processor stack. The data can be either an immediate byte or an immediate word. If the data is a byte, it will be sign extended to a word before it is pushed onto the stack (since all stack operations are word operations).

• PUSHA, POPA

These instructions allow all of the general purpose 80186 registers to be saved on the stack, or restored from the stack. The registers saved by this instruction (in the order they are pushed onto the stack) are AX, CX, DX, BX, SP, BP, SI, and DI. The SP value pushed onto the stack is the value of the register before the first PUSH (AX) is performed; the value popped for the SP register is ignored.

This instruction does not save any of the segment registers (CS, DS, SS, ES), the instruction pointer (IP), the flag register, or any of the integrated peripheral registers.

• IMUL by an immediate value

This instruction allows a value to be multiplied by an immediate value. The result of this operation is 16 bits long. One operand for this instruction is obtained using one of the 80186 addressing modes (meaning it can be in a register or in memory). The immediate value can be either a byte or a word, but will be sign extended if it is a byte. The 16-bit result of the multiplication can be placed in any of the 80186 general purpose or pointer registers.

This instruction requires three operands: the register in which the result is to be placed, the immediate value, and the second operand. Again, this second operand can be any of the 80186 general purpose registers or a specified memory location.

• shifts/rotates by an immediate value

The 80186 can perform multiple bit shifts or rotates where the number of bits to be shifted is specified by an

immediate value. This is different from the 8086, where only a single bit shift can be performed, or a multiple shift can be performed where the number of bits to be shifted is specified in the CL register.

All of the shift/rotate instructions of the 80186 allow the number of bits shifted to be specified by an immediate value. Like all multiple bit shift operations performed by the 80186, the number of bits shifted is the number of bits specified modulus 32 (i.e. the maximum number of bits shifted by the 80186 multiple bit shifts is 31).

These instructions require two operands: the operand to be shifted (which may be a register or a memory location specified by any of the 80186 addressing modes) and the number of bits to be shifted.

• block input/output

The 80186 adds two new input/output instructions: INS and OUTS. These instructions perform block input or output operations. They operate similarly to the string move instructions of the processor.

The INS instruction performs block input from an I/O port to memory. The I/O address is specified by the DX register; the memory location is pointed to by the DI register. After the operation is performed, the DI register is adjusted by 1 (if a byte input is specified) or by 2 (if a word input is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The ES segment register is used for memory addressing, and cannot be overridden. When preceded by a REPEAT prefix, this instruction allows blocks of data to be moved from an I/O address to a block of memory. Note that the I/O address in the DX register is not modified by this operation.

The OUTS instruction performs block output from memory to an I/O port. The I/O address is specified by the DX register; the memory location is pointed to by the SI register. After the operation is performed, the SI register is adjusted by 1 (if a byte output is specified) or by 2 (if a word output is specified). The adjustment is either an increment or a decrement, as determined by the Direction bit in the flag register of the processor. The DS segment register is used for memory addressing, but can be overridden by using a segment override prefix. When preceded by a REPEAT prefix, this instruction allows blocks of data to be moved from a block of memory to an I/O address. Again note that the I/O address in the DX register is not modified by this operation.

Like the string move instruction, these two instructions require two operands to specify whether word or byte operations are to take place. Additionally, this determination can be supplied by the mnemonic itself by adding a "B" or "W" to the basic mnemonic, for example:

INSB	; perform byte input
REP OUTSW	; perform word block output

• BOUND

The 80186 supplies a BOUND instruction to facilitate bound checking of arrays. In this instruction, the calculated index into the array is placed in one of the general purpose registers of the 80186. Located in two adjacent word memory locations are the lower and upper bounds for the array index. The BOUND instruction compares the register contents to the memory locations, and if the value in the register is not between the values in the memory locations, an interrupt type 5 is generated. The comparisons performed are SIGNED comparisons. A register value equal to either the upper bound or the lower bound will not cause an interrupt.

This instruction requires two arguments: the register in which the calculated array index is placed, and the word memory location which contains the lower bound of the array (which can be specified by any of the 80186 memory addressing modes). The memory location containing the upper bound of the array must follow immediately the memory location containing the lower bound of the array.

• ENTER and LEAVE

The 80186 contains two instructions which are used to build and tear down stack frames of higher level, block structured languages. The instruction used to build these stack frames is the ENTER instruction. The algorithm for this instruction is:

```
PUSH BP          /* save the previous frame pointer */
if level = 0 then
  BP := SP;
else
  temp1 := SP; /* save current frame pointer */
```

```
temp2 := level - 1;
do while temp2 > 0 /* copy down previous level frame */
  BP := BP - 2; /* pointers */
  PUSH [BP];
  BP := temp1;
  PUSH BP; /* put current level frame pointer */

/* in the save area */
SP := SP - disp; /* create space on the stack for */
```

```
/* local variables */
```

Figure H-1 shows the layout of the stack before and after this operation.

This instruction requires two operands: the first value (disp) specifies the number of bytes the local variables of this routine require. This is an unsigned value and can be as large as 65535. The second value (level) is an unsigned value which specifies the level of the procedure. It can be as great as 255.

The 80186 includes the LEAVE instruction to tear down stack frames built up by the ENTER instruction. As can be seen from the layout of the stack left by the ENTER instruction, this involves only moving the contents of the BP register to the SP register, and popping the old BP value from the stack.

Neither the ENTER nor the LEAVE instructions save any of the 80186 general purpose registers. If they must be saved, this must be done in addition to the ENTER and the LEAVE. In addition, the LEAVE instruction does not perform a return from a subroutine. If this is desired, the LEAVE instruction must be explicitly followed by the RET instruction.

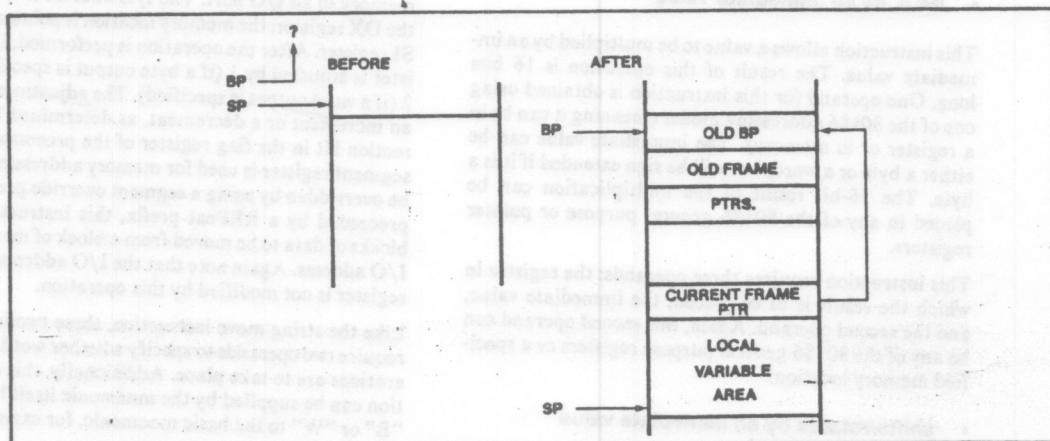


Figure H-1. ENTER Instruction Stack Frame

APPENDIX I: 80186/80188 DIFFERENCES

The 80188 is exactly like the 80186, except it has an 8 bit external bus. It shares the same execution unit, timers, peripheral control block, interrupt controller, chip select, and DMA logic. The differences between the two caused by the narrower data bus are:

- The 80188 has a 4 byte prefetch queue, rather than the 6 byte prefetch queue present on the 80186. The reason for this is since the 80188 fetches opcodes one byte at a time, the number of bus cycles required to fill the smaller queue of the 80188 is actually greater than the number of bus cycles required to fill the queue of the 80186. As a result, a smaller queue is required to prevent an inordinate number of bus cycles being wasted by prefetching opcodes to be discarded during a jump.
- AD8-AD15 on the 80186 are transformed to A8-A15 on the 80188. Valid address information is present on these lines throughout the bus cycle of the 80188. Valid address information is not guaranteed on these lines during idle T states.
- $\overline{\text{BHE}}/\text{S7}$ is always defined HIGH by the 80188, since the upper half of the data bus is non-existent.

- The DMA controller of the 80188 only performs byte transfers. The B/W bit in the DMA control word is ignored.
- Execution times for many memory access instructions are increased because the memory access must be funnelled through a narrower data bus. The 80188 also will be more bus limited than the 80186 (that is, the execution unit will be required to wait for the opcode information to be fetched more often) because the data bus is narrower. The execution time within the processor, however, has not changed between the 80186 and the 80188.

Another important point is that the 80188 internally is a 16-bit machine. This means that any access to the integrated peripheral registers of the 80188 will be done in 16-bit chunks, NOT in 8-bit chunks. All internal peripheral registers are still 16-bits wide, and only a single read or write is required to access the registers. When an access is made to the internal registers, only a single bus cycle will be run, and only the lower 8-bits of the written data will be driven on the external bus. All accesses to registers within the integrated peripheral block must be WORD accesses.

